UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Adaptación del Stanford Parser al español

Borja Colmenarejo García
Tutor: Pablo Alfonso Haya Coll

Junio 2016

Adaptación del Stanford Parser al español

AUTOR: Borja Colmenarejo García
TUTOR: Pablo Alfonso Haya Coll

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2016

Resume

El presente Trabajo Fin de Grado tiene como objetivo entender y mejorar la adaptación del analizador sintáctico realizado por la Universidad de Stanford para la lengua española. Este software es entrenado con un modelo que analiza y clasifica frases a partir del análisis más probable. Su cometido es analizar las sentencias proporcionadas para construir la estructura interna de la oración mediante agrupación de términos en sintagmas, clausulas y oraciones.

En el documento se explica la necesidad de la creación de herramientas para subsanar una de las limitaciones principales que carece el software, como es la validación y obtención de rendimiento de los análisis realizados.

El analizador sintáctico en el que se centra este trabajo consiste en un clasificador que tiene que ser entrenado mediante un conjunto de frases de las que se determina el resultado del análisis sintáctico. En esta línea se trabaja con la hipótesis de que un mayor entrenamiento se verá reflejado en mejor rendimiento. Por otra parte, se afinan las etiquetas utilizadas en la clasificación previa de datos utilizados para el entrenamiento, consiguiendo así un mejor rendimiento.

Además, para mejorar los datos de entrenamiento se ha desarrollado una herramienta específica para aumentar el número de datos de forma artificial y estudiar la hipótesis con la que se trabaja.

Aunque en este trabajo se llegan a varias conclusiones sobre el funcionamiento del software y del trato que realiza de las etiquetas, se puede seguir trabajando. De manera que este proyecto abre las puertas a futuras investigaciones, aportando información sobre cómo mejorar la adaptación al español a partir de los archivos empleados para la ejecución del software.

Palabras clave

Stanford Parser, lingüística computacional, treebank, corpus y analizador sintáctico.

Abstract

This Bachelor Thesis has the goal to understand and develop the Spanish version of the parser starring by Stanford University. This software is based on a model that analyzes and classifies sentences from the most likelihood analysis. Software´s main role is to parse given sentences to build its internal structure distinguishing between syntagmas, clause and phrase.

The following essay discuss the need of creating new main tools to overcome the limitations that, nowadays, are in the software. The main limitations that this version of the software host are the validation and extraction of the performance from test results.

The parser, which this paper is focused, consist on a classifier that has to be trained by a group of sentences, from which the result of the syntactic analysis comes. The hypothesis under this line of work stays that a higher training leads to a mayor efficiency.

On the other hand, we tune the tags used in the previous classification of data that has been used for the training, achieving a better performance.

Moreover, to improve the train data there has been developed a specific tool to increase the number of artificial data and to study the stated hypothesis.

Finally, going through the main conclusions carried by this thesis on the performance of the mentioned software and over the treatment of the tags; we can observe that they can be key matters to future research projects. Providing more information in the development of the Spanish version, it will adjust these thesis conclusions themselves and so, it would allow go further in the improvement of the Stanford Parser.

Keywords

Stanford Parser, computational linguistics, treebank, corpus and parser.

Agradecimientos

A mi familia y amigos por todo su apoyo durante todos estos años. A mis compañeros del grado que tanto hemos luchado y sufrido juntos, y especialmente a mi tutor por todos sus consejos durante el desarrollo del proyecto.

Sin olvidar mencionar a Laura, por su paciencia y ayuda.

A todos ¡Muchas gracias!

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICAS

ÍNDICE DE TABLAS

1 Introducción

El Trabajo de Fin de Grado "La adaptación del Stanford Parser al español" nace de una idea del Departamento de Lingüística conjunta con el Departamento de Informática de la Universidad Autónoma de Madrid, para intentar mejorar el analizador sintáctico de la Universidad de Stanford (Stanford University) con el fin de tratar la lengua española con una computadora.

Para ello se plantearon dos líneas de trabajo, la primera consistió en incluir nuevas funcionalidades al software y la segunda en proporcionar herramientas para mejorar dicha adaptación.

La primera se descartó por ser un proyecto muy extenso con multitud de clases, en ocasiones mal documentadas, y se trabajó en la segunda línea. De esta forma el trabajo está centrado en la creación de herramientas y en mejorar los ficheros necesarios para la ejecución del software.

1.1 Motivación

Este proyecto parte de la motivación por aprender e investigar en profundidad sobre las lenguas humanas, en concreto sobre la lengua española. La lengua española presenta una gran riqueza en léxico y estructuras gramaticales por lo que es un reto importante adaptar una herramienta que ha sido probada en lengua inglesa, al español.

Para la realización del trabajo se emplea el analizador gramatical desarrollado por la Universidad de Stanford (Stanford Parser en el resto del documento). Este programa es un proyecto de código abierto, por lo cual incita a participar y aportar en la comunidad.

El software en su adaptación para el español es el que menos desarrollado se encuentra, después del árabe, así que el reto se supone aun mayor, al ser una de las partes que más recorrido tiene y de las que a priori se ha trabajado menos.

1.2 Objetivos

El objetivo principal del proyecto es adaptar la funcionalidad del Stanford Parser para incorporar información lingüística del español que permita desambiguar, de una manera más eficiente, los distintos tipos de verbos, nombres y adjetivos, lo que origina la mejora de la certeza y precisión en los análisis. Siguiendo esta línea se pretende conseguir entender el funcionamiento del Stanford Parser en su adaptación al español para mejorar la misma. De modo que se tenga certeza de cómo se puede mejorar y porqué.

Uno de los problemas más destacados que presenta el programa es la carencia de un sistema de evaluación. Por lo que se persigue la creación de una herramienta para solucionar dicha limitación. Del mismo modo se quiere aportar a la comunidad del proyecto la herramienta para continuar con su desarrollo y seguir mejorándola.

Por último, se pretende crear una herramienta específica para el laboratorio de lingüística computacional, mediante la cual se puedan generar frases semiautomáticas a partir de unas frases dadas.

Ambas herramientas se desarrollarán para tener la mayor usabilidad posible, pues se pretende que cualquier persona sea capaz de usarlas. También se implementaran flexibles para que cualquier desarrollador las adapte a sus necesidades, cree nuevas funcionalidades o nuevas herramientas a partir de las entregadas.

1.3 Organización de la memoria

La estructura del presente documento consta de los siguientes capítulos:

- **Estado del arte** se introducen una serie de conceptos clave y el marco actual en el que esta agregado este proyecto.
- **Diseño** se proporciona un diseño de las herramientas implementadas durante el proyecto.
- **Desarrollo** se explican las estructuras de las herramientas y los algoritmos más relevantes que las mismas utilizan.
- **Integración, pruebas y resultados** se muestran las pruebas que certifican las herramientas creadas.
- **Experimentos y resultados** se muestran los experimentos que se han llevado a cabo siguiendo la línea de trabajo expuesta en el proyecto, así como las hipótesis en las que se basa.

2 Estado del arte

En este apartado se pretende situar al lector en la esencia de los conceptos que más adelante serán utilizados, se hablará de varios términos clave como son analizador sintáctico y *Penn Treebank* o *Corpus*. Finalmente se tratará la situación actual del software en el cual se centra el proyecto, el Stanford Parser, explicando que problemas presenta la adaptación al español.

2.1 Lingüística Computacional

La lingüística computacional es un área de investigación que se basa en el procesamiento de lenguajes naturales. Se considera una rama de la lingüística que mezcla el análisis sintáctico y morfológico con herramientas informáticas para el estudio de la lengua humana hablada y escrita.

Desde el ámbito informático es considerada una subdisciplina de la inteligencia artificial que intenta entender mejor el lenguaje humano, a través de algoritmos de análisis sintáctico y matemáticos basados en probabilidad.

El principal problema que presenta la lengua humana es la denominada *infinitud discreta* por Chomsky. La *infinitud discreta* está muy ligada a la recursión, pues es un elemento necesario para cualquier teoría humana que trata de formalizar oraciones a partir de un número limitado de elementos constituyentes de un lenguaje. Según el propio **Noam Chomsky**:

> *(…) se pueda construir con un par de docenas de sonidos una infinitud de expresiones que nos permiten revelar a otros lo que pensamos, imaginamos y sentimos.* (***Chomsky***, 1998).

Desde hace varias décadas la lingüística computacional está en pleno desarrollo, y parece un campo muy interesante donde investigar pero que presenta un problema esencial muy difícil de resolver pues la infinitud discreta no es fácil de abordar desde la computación actual. **Grigory Sidorov** ya exponía esto:

> *En general, la lingüística computacional es una ciencia que tiene por el momento más problemas que soluciones, pero es un campo de investigación muy interesante y prospectivo.* (***Sidorov**, 2001*).

2.2 Analizadores sintácticos

Análisis sintáctico o *parsing* es el proceso de análisis de cadenas de símbolos, escritos en lenguaje natural o lenguaje máquina, empleando las reglas de una gramática formal. En los campos de lingüística y de la informática puede tener un significado diferente.

Por lo que, hay que aclarar que este término es siempre tratado desde el ámbito lingüístico en este proyecto, el cual consiste en entender el significado riguroso de la frase.

Un analizador sintáctico es un software basado en algoritmos de *parsing*, algoritmos probabilísticos y una combinación de ambos, que convierte un texto de entrada en otras estructuras. En otras palabras, es un analizador sintáctico de una lengua natural.

En 2012 ya se explicaban los fundamentos de los algoritmos tratándolos como paradigmas de la investigación:

> *A lo largo de la historia del procesamiento del lenguaje natural los dos enfoques principales de investigación adoptados han sido los paradigmas simbólico y estadístico:*
>
> *(i). El enfoque simbólico se caracteriza por la construcción de sistemas que almacenan explícitamente los hechos lingüísticos (…).*
>
> *(ii). El enfoque estadístico se caracteriza por la construcción de sistemas que no almacenan explícitamente el conocimiento lingüístico o del mundo, sino que aplican técnicas matemáticas sobre extensos córpora[1] informatizados con el fin de inferir dicho conocimiento. (**Periñan-Pascual**, 2012).*

Actualmente los analizadores sintácticos están más avanzados que cuando aparecieron por primera vez, pues el avance de los ordenadores y de la computación en general ha dejado mejorar los rendimientos de los algoritmos y por tanto el análisis se realiza más rápido. También, con el desarrollo de la computación, aparecen nuevas formas de procesar datos permitiendo crear algoritmos combinatorios. Por lo que, se puede decir que los analizadores sintácticos seguirán evolucionando con la informática manteniéndose actualizados y permitiendo nuevas formar de análisis sintáctico del lenguaje.

2.2.1 Niveles de análisis lingüístico

En lingüística existen varios niveles de análisis, aunque no todos los lingüistas están de acuerdo, la mayoría coinciden en al menos los siguientes: fonético, fonológico, morfológico, sintáctico y semántico. Si bien estos niveles están conectados entre ellos no es necesario que aparezcan todos para hacer un análisis lingüístico correcto. Para entender los diferentes tipos análisis, a continuación, se explica el significado de cada uno de ellos:

- Fonética: estudia los sonidos producidos por los seres humanos para comunicarse unos con otros. El estudio de estos sonidos provee las herramientas teóricas para reconocer, pronunciar y escribir cualquier sonido del habla.

[1] Córpora, traducción libre de *corpus*, también conocido como *Penn Treebank*, o solo *treebank*. (veases 2.2.2).

- Fonología: describe el modo en que los sonidos funcionan. Establece las bases para la relación de los sonidos producidos por los seres humanos y la forma de emparejar, asociar y emplear los sonidos.

- Morfología: estudia la estructura interna de las palabras para delimitar, definir y clasificar sus unidades y las clases de palabras que forma y puede llegar a formar.

- Sintaxis: es la rama de la lingüística dedicada al análisis sintáctico. Estudia las reglas, principios y normas que rigen la combinatoria de las palabras o secuencias de palabras que forman sintagmas y oraciones.

- Semántica: Estudia el significado de las expresiones dentro de las frases. Se refiere a los aspectos del significado, sentido o interpretación de los signos lingüísticos.

2.3 Penn Treebank

Penn Treebank o *corpus* es un archivo de frases donde cada oración es representada con una misma estructura, normalmente árboles, y que es previamente anotada ajustándose a la gramática formal empleada.

UAM Spanish Treebank, se trata de un corpus de oraciones extraídas de periódicos con un total de 1600 frases seleccionadas, anotadas, revisadas y depuradas por los lingüistas miembros del proyecto Antonio Moreno, Susana López y Manuel Alcántara, y gracias a la ayuda de los lingüistas computacionales Fernando Sánchez y Ralph Grishman en la aportación de herramientas para la anotación y la depuración.

Para el desarrollo de este Trabajo Final de Grado se usa una versión reducida del UAM Spanish Treebank. En dicha versión se han quitado anotaciones que para este caso no tenían relevancia, dejando tan solo las anotaciones que denotan la función que desempeña la palabra dentro de la frase, sujeto, complemento, objeto, sintagma, etc., y la categoría sintáctica de palabra, sustantivo, verbo, preposición, etc.

Este *treebank* se encuentra en continua evolución siendo el objetivo del proyecto llegar a las 5000 frases realizando el trabajo de construcción de manera semiautomática. Toda la información se detalla en la página web[2] oficial del *corpus* original.

2.4 Stanford Parser

El Stanford Parser es un analizador sintáctico probabilístico desarrollado por la Universidad de Stanford que se basa en seleccionar el mejor análisis según aquel que sea más probable a partir de un conjunto de ejemplos analizados correctamente por lingüistas.

[2] http://www.lllf.uam.es/ESP/Treebank.html#tools

El software es, en esencia, un clasificador que separa un texto en términos y no términos. En algoritmos de aprendizaje automático los términos son los llamados datos de entrenamiento, mientras que los no términos son datos irrelevantes que no afectan al clasificador, por lo que el Stanford Parser desecha estos datos. Es muy importante, por ello, que los términos sean anotados por expertos de la materia.

Actualmente la mayor parte de los analizadores sintácticos modernos emplean el enfoque de algoritmo combinatorio, mientras el Stanford Parser está basado en un ámbito puramente probabilístico para hacer la división de términos. Por lo que, necesita ser entrenado con un *Corpus* o *Penn Treebank* previamente anotado con información morfológica. Esto lo explican **Antonio Moreno Sandoval** y **Leonardo Campillos Llanos**:

> *Las aproximaciones basadas en aprendizaje automático son un tipo de estrategia estadística que consiste en entrenar los programas con los datos de un corpus previamente anotado con términos por especialistas.* ***(Moreno-Sandoval y Campillos-Llanos****, 2015)*

El uso de algoritmos probabilísticos hace del software una herramienta complicada en el uso, pues emplea multitud de parámetros para intentar ajustar el análisis en su ejecución.

Además al emplear este enfoque probabilístico, se realiza un análisis ligero, es decir, se centra solo en los núcleos principales de la oración, el verbo y los sustantivos. En este tipo de análisis solo se tiene en cuenta el nivel de análisis sintáctico de la lingüística.

Los sustantivos desempeñan funciones muy diversas dentro de una frase, pero para este caso las funciones destacadas en las que se centra el analizador sintáctico son las de sujeto y objetos directo e indirecto, en otras palabras, las funciones inmediatamente dependientes del verbo. Lo que concluye que el verbo es el elemento principal de la oración, el sujeto y los objetos los elementos secundarios y relevando al resto de componentes a un plano más lejano.

Para realizar un análisis sintáctico exhaustivo de oraciones es esencial la anotación del lema del verbo, dotando así al analizador sintáctico de información relevante para la elección y asignación de elementos secundarios. El lema en lingüística es una palabra que forma una unidad semántica y que puede constituir una entrada de diccionario.

El Stanford Parser trabaja con la dependencia del software coreNPL de la Universidad de Stanford. Este elemento proporciona un conjunto de herramientas de análisis lingüístico. La incorporación de esta dependencia al Stanford Parser proporciona, entre otras, las anotaciones que soporta cada lenguaje humano. Al ser una herramienta orientada principalmente al inglés, las anotaciones para otros lenguajes no están del todo desarrolladas, siendo el español el que menos anotaciones incorpora después del árabe. Una de las grandes carencias de estas anotaciones es la del lema (*Figura 1*).

ANNOTATOR	AR	ZH	EN	FR	DE	ES
Tokenize / Segment	✓	✓	✓	✓		✓
Sentence Split	✓	✓	✓	✓	✓	✓
Part of Speech	✓	✓	✓	✓	✓	✓
Lemma			✓			
Named Entities		✓	✓		✓	✓
Constituency Parsing	✓	✓	✓	✓	✓	✓
Dependency Parsing		✓	✓	✓	✓	
Sentiment Analysis			✓			
Mention Detection		✓	✓			
Coreference		✓	✓			
Open IE			✓			

***Figura 1*. Anotaciones soportadas para los diferentes lenguajes humanos por el coreNLP. Fuente (*Stanford CoreNLP – a suite of core NLP tolos | Stanford CoreNLP 2016*).**

A día de hoy, el software está desarrollado completamente para el lenguaje para el cual fue pensado en un primer momento, el inglés, en cuanto al resto de adaptaciones todavía les queda mucha trayectoria, sobre todo en la incorporación de anotaciones que poco a poco seguramente irán apareciendo en versiones futuras del proyecto.

3 Diseño

En este apartado se presentan el diseño de las dos herramientas desarrolladas durante el Trabajo Final de Grado. Al tratar el software del Stanford Parser como una caja negra, el desarrollo de las herramientas está pensado para manejarse mediante ficheros externos almacenados de forma temporal.

3.1 Plataforma de evaluación

La limitación que presenta el programa en cuanto a un sistema de evaluación que permita obtener un rendimiento del mismo se subsana creando una plataforma de evaluación en cuatro etapas: generación de ficheros, entrenamiento, test y obtención de rendimiento.

La evaluación hace referencia al rendimiento obtenido por el software al ser entrenado con un *treebank* concreto. Esta plataforma permite obtener tanto una validación cruzada como una validación simple partiendo de un solo *treebank*. La validación cruzada consiste en entrenar el Stanford Parser con un conjunto de datos que no contenga ninguna da las frases del conjunto de validación mientras que en la validación simple emplea todas las frases contenidas en el conjunto de entrenamiento y valida las mismas. En el desarrollo se explica cómo ha sido posible crear esta adaptación.

Independientemente del tipo de validación, ésta se basa en la creación de ficheros externos que se van obteniendo y utilizando en las diferentes fases cuando son necesarios. En la *Figura 2* se muestra el proceso completo, diferenciando cada fase con un color. La etapa de generación de ficheros en azul, entrenamiento en naranja, test en verde y obtención de rendimiento en morado.

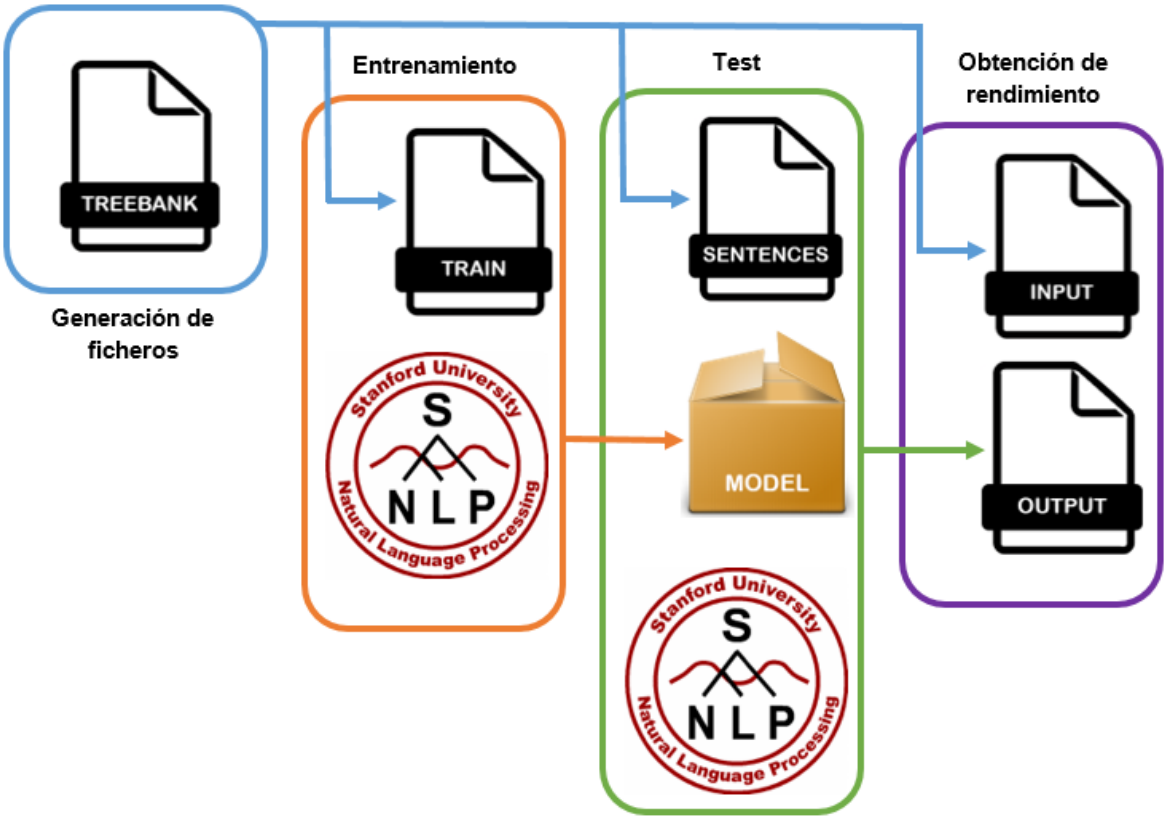


***Figura 2*. Etapas de plataforma de evaluación del Stanford Parser.**

Las cuatro etapas merecen ser explicadas individualmente en los siguientes apartados.

3.1.1 Generación de ficheros

La primera fase es generar los ficheros principales que serán empleados en las distintas etapas del proceso de evaluación. A partir del *corpus*, la herramienta lee en memoria los árboles que lo constituyen y genera tres conjuntos de ficheros: ficheros *train*, ficheros *input* y ficheros *sentences*.

El fichero *train* contiene las frases en forma de árbol sintáctico con las que el Stanford Parser será entrenado en la siguiente etapa (*Figura 3*). Para mantener los arboles balanceados correctamente se emplea el formato *lisp* para el fichero. Las oraciones que conforman este fichero será un conjunto extraído del *treebank* original las cuales solo se usaran para esta función.

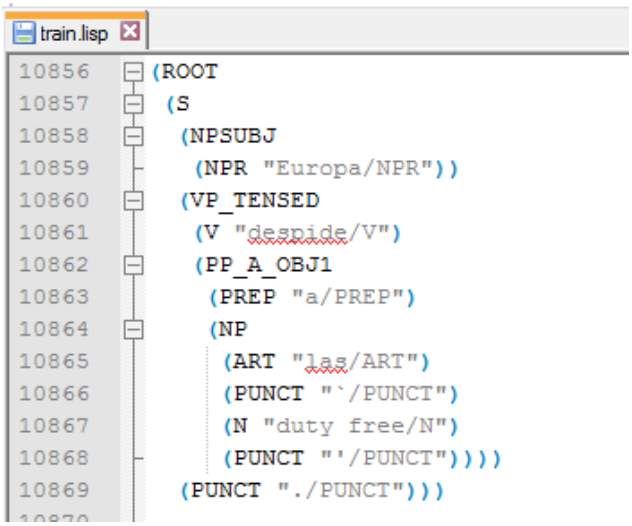


***Figura 3*. Ejemplo del ficher de *train*.**

El fichero *input* contiene las frases en forma de árbol bien etiquetadas, para una vez obtenido un fichero con las frases etiquetadas realizar la comparación y obtener el rendimiento (***Figura** 4*). El fichero estará compuesto por las frases que se quieren validar, es decir, con las frases que no estén contenidas en el fichero *train*. El formato del fichero debe ser texto plano, por lo que se escogió *txt*.

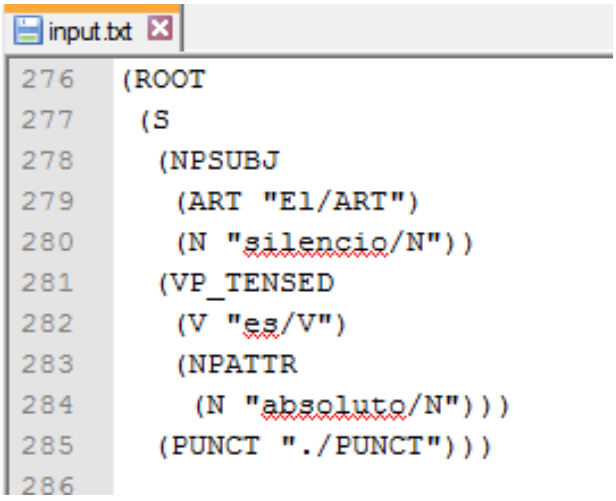


***Figura 4*. Ejemplo de fichero de *input*.**

El fichero *sentences* contiene las oraciones que serán etiquetadas (***Figura** 5*). La etiquetación de la frase consiste en crear el árbol del análisis sintáctico de la oración, de forma que, ayudando con la categoría gramatical (artículo, sustantivo, verbo, etc.,) agrupa las palabras en sintagmas para crear el árbol. El contenido de este fichero será el mismo contenido del fichero *input*, teniendo en cuanta que el anterior almacena las frases en forma de árbol, y este en forma de oración.

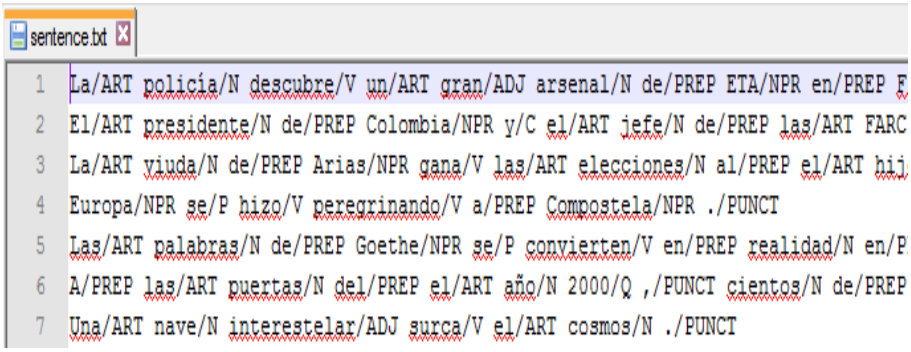


***Figura 5*. Ejemplo de fichero de *sentences*.**

Para la realización de la validación cruzada hay que tener en cuenta que en cada iteración las frases de *input* y *sentences* deben coincidir, y además que dentro de las frases de *train* no aparecerán. Si tenemos *n* particiones, *n-1* conjuntos se dedicarán al *train*, mientras que el conjunto restante se dedicará al *test*, siendo este conjunto el que coincidirá en los ficheros de *input* y *sentences*.

3.1.2 Entrenamiento

Con los ficheros de *train* generados en la etapa anterior se entrena el Stanford Parser. El resultado de este entrenamiento es un archivo comprimido que contiene el *modelo* con el cual se realizara el test en la siguiente fase.

Archivo del *modelo* es un diccionario de palabras, etiquetas y estructuras generado a partir de un previo entrenamiento del Stanford Parser.

3.1.3 Test

La herramienta ejecuta el Stanford Parser, el cual es configurado con el archivo *modelo*, y analiza las frases contenidas en el fichero *sentences*, generando un nuevo fichero, *output*, que contiene las frases etiquetadas con el resultado del análisis sintáctico.

El fichero *output* contiene las frases etiquetadas por el Stanford Parser basándose en el modelo de entrenamiento. El formato de salida de este archivo es *txt* al igual que el fichero *input*, ya que es necesario un fichero en texto plano para realizar la comparación de etiquetas y la posterior obtención de rendimiento. Las frases que componen este archivo son las mismas frases contenidas en el fichero *sentences* pero en forma de árbol y con las etiquetas creadas por el Stanford Parser.

3.1.4 Obtención de rendimiento

El rendimiento se realiza comparando las etiquetas de las frases. Para ello es necesario tener dos ficheros, el fichero *input* y el fichero *output*, con las mismas frases etiquetadas y, únicamente, comparar las etiquetas de los árboles. Una vez que tenemos los ficheros se obtienen los resultados de la comparación expresados como *true positives (tp)*, *false positives (fp)* y *false negatives (fn)*.

True positive, se refiere a la etiqueta que se encuentran en ambos archivos en la misma posición.

False positives, se trata de la etiqueta que debería de estar en un lugar en ambos ficheros y que solo se encuentra en el fichero obtenido tras la ejecución del test (*output*). En otras palabras, son etiquetas que están de más en el fichero de salida.

False negatives, la diferencia entre el número de etiquetas correctas y las etiquetas no catalogadas, es decir, son etiquetas que deberían aparecer en el fichero de salida pero que no están.

Tanto los *false negatives* como los *false positives* hace referencia al número de etiquetas, no solo a la etiqueta en sí.

Con estos datos se procede a hallar la cobertura (*recall*) y la precisión (*precisión*) del modelo en función del valor F.

La elección de esta medida se justifica con los autores Antonio Moreno Sandoval y Leonardo Campillos Llanos:

> *En Lingüística Computacional y concretamente en el campo de la recuperación de información se emplea de manera generalizada tres medidas de evaluación: precisión, cobertura (recall) y la medida F.* ***(Moreno-Sandoval y Campillos-Llanos****, 2015)*

El valor F, conocido como *F1Score*, se considera una medida armónica que combina los valores de la precisión y la exhaustividad del modelo, donde alcanza su mejor valor en 1 y el peor en 0.

La *precision* es la relacion de datos recuperados que son relevantes, dando respuesta a la pregunta ¿Cuántos de las datos son relevantes?. Calculada mediante la siguiente igualdad:

$$precision = \frac{tp}{tp + fp}$$

El *recall* es la relación de datos relevantes que han sido recuperadas, respondiendo a la pregunta ¿Cuántos de los datos seleccionados son relevantes?. La siguiente igualdad muestra como calcularla:

$$recall = \frac{tp}{tp + fn}$$

La medida F es la relación que presenta la *precision* con el *recall* de modo que la relación de ambas se haga a partes iguales dando la misma importancia a ambos factores. La siguiente igualdad muestra cómo se relacionan los factores:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

3.2 Sistema de generación de frases

Para incrementar el número de frases del *corpus* y siguiendo las premisas del proyecto del *treebank* original es necesario un proceso de generación de frases que automatice el proceso. Para la generación de frases se plantearon múltiples estrategias: sustituir los verbos por sinónimos, combinar diferentes predicados, fijar una estructura de sintagmas en una oración y rellanar cada sintagma con uno apropiado en cada caso, combinar seleccionando un solo sintagma, etc.

En este caso se ha creado un sistema específico que combina las diferentes frases utilizando la estrategia de combinar un solo tipo de sintagma para obtener nuevas frases. Con este método se optó por recombinar los sintagmas nominales sujeto de las oraciones etiquetados con cualquier tipo de sintagma nominal. Lo que suponía un reto, pues al realizar esta recombinación se quería mantener la concordancia de número[3] de las oraciones entre el sujeto y el verbo que lo acompaña para evitar enseñar erróneamente al Stanford Parser.

Para solucionar el problema de concordancia había que utilizar una nueva etiqueta que indicase el número de los nodos etiquetados como sintagmas nominales, para ello se empleó el *treebank* original con todas las anotaciones haciendo uso de las etiquetas necesarias. De tal forma que se pueden distinguir dos grupos de sintagmas nominales: los sintagmas en singular y los sintagmas en plural.

Los ficheros externos necesarios para esta herramienta son dos, el *treebank* original con todas las anotaciones y un fichero que contenga las frases para recombinar.

Una vez seleccionadas las frases[4] y leídos los sintagmas, diferenciando el grupo de los que son en singular y los que son en plural, se selecciona un sintagma del grupo adecuado de manera aleatoria y se sustituye por el sujeto de la oración de modo que se crea una oración con estructura similar, pero con diferente sujeto. Por ejemplo:

La frase original podría ser:

"Informó el presidente que no habrá aumentos de sueldo."

[3] En esta subsección cuando se habla de número se refiere al número gramatical de manera generalizada, es decir, singular o plural.

[4] Para hacer esta selección hay que fijarse en la estructura de la frase y no en la oración ni el significado de la misma.

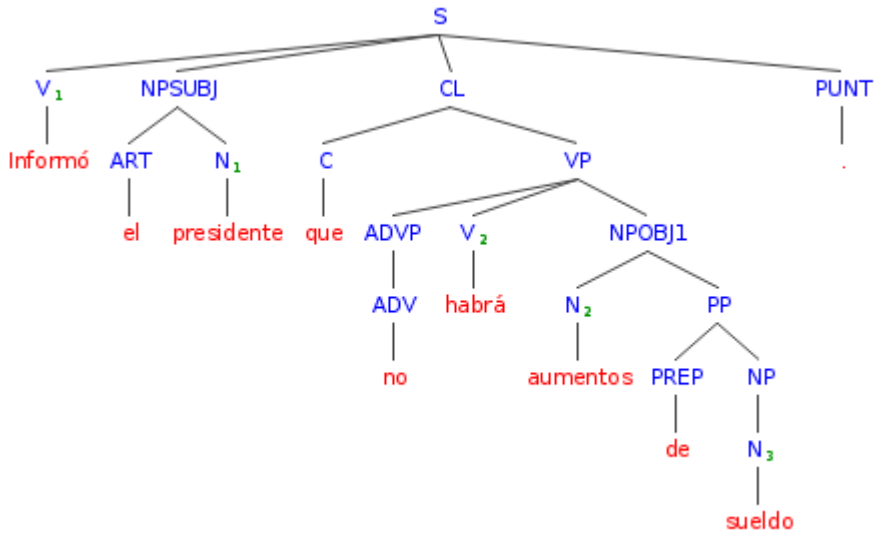


Figura 6. **Representación en forma de árbol de la frase original de ejemplo.**

Tal y como se muestra en el análisis "*el presidente*" seria el sujeto, por lo que para crear una nueva frase tan solo habría que sustituirlo por un sintagma nominal y marcar al nuevo sintagma nominal como sujeto, por ejemplo, en este caso por "*Madrid*" de este modo la nueva frase quedaría como sigue:

"Informó Madrid que no habrá aumentos de sueldo."

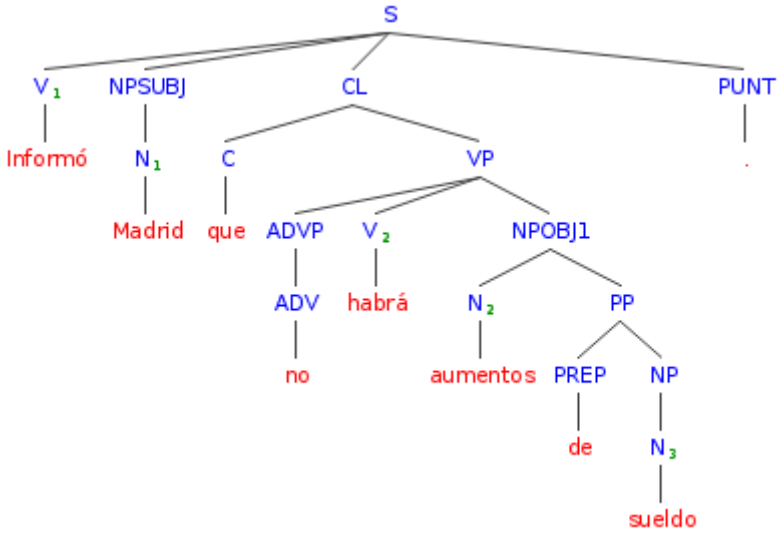


Figura 7. **Representación en forma de árbol de la nueva frase generada a partir del ejemplo.**

La generación de frases se hace para entrenar una determinada faceta del lenguaje natural español. Para ello se eligen que tipo de estructuras de oración se quieren mantener y se recombinan a partir de esas estructuras, obteniendo de esta forma nuevas frases con estructura similar.

4 Desarrollo

En esta sección se explica la implementación de las herramientas creadas en el Trabajo Final de Grado. Las herramientas han sido desarrolladas en *java*. Se ha creado una clase abstracta que engloba las funciones básicas que se emplearan en las diferentes herramientas, de modo que como se muestran en los diagramas de clases de cada herramienta, para el desarrollo de las mismas es necesario extender la clase para utilizar las funciones que alberga. La función principal de clase es leer los árboles en memoria para su posterior uso. Para hacer flexible la lectura de los árboles se ha creado una función que debe ser implementada según las necesidades de cada herramienta.

La principal diferencia que existe en esta función para las dos herramientas se debe al *treebank* que en cada momento se lee. La diferencia en sí, es la incorporación o no del número del sintagma. De forma que lo incorporamos cuando se quieren generar nuevas frases, para, como ya se explica en la sección anterior (véase 3.2), mantener la concordancia entre el verbo y los sintagmas nominales dependientes del mismo.

4.1 Software de validación

La herramienta de validación se ajusta al siguiente diagrama de clases:

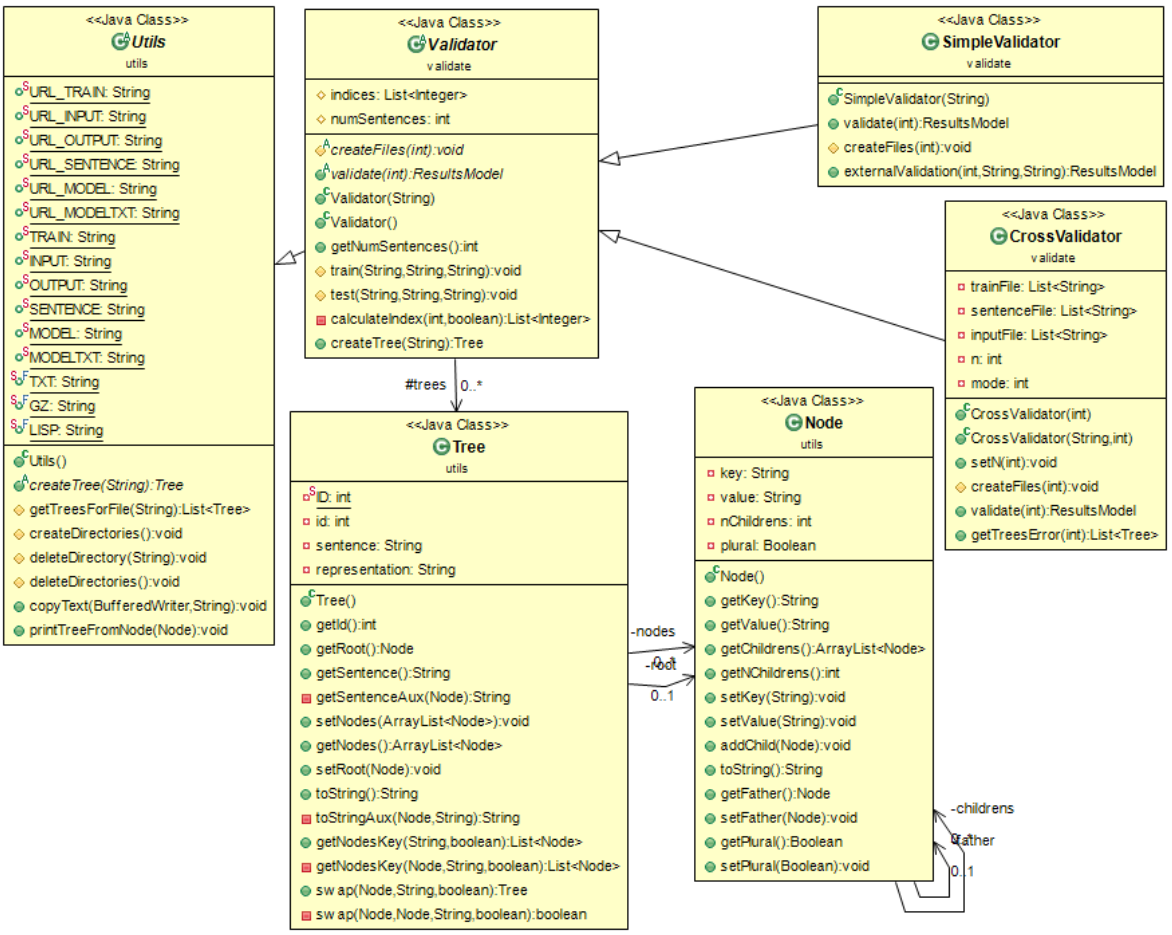


***Figura 8*. Diagrama de clases de la herramienta de validación.**

Como se muestra en el diagrama, se ha creado una clase abstracta de la cual heredan los diferentes modos de evaluar, para proporcionar flexibilidad al diseño y poder incorporar nuevas formas de validar. Gracias a esta clase es posible la adaptación del validador a los diferentes métodos, validación cruzada y validación simple, tan sólo es necesario la implementación de dos funciones, la función de evaluar (*validate*) y la función de generación de ficheros (*createFiles*).

Además, esta decisión se tomó pensando en aportar a la comunidad del proyecto una herramienta desde la cual se pudiese seguir mejorando y trabajando en el analizador sintáctico.

En esta herramienta cabe destacar la importancia de dos algoritmos, el algoritmo de verificación y el algoritmo de creación de árboles, que serán detallados en las siguientes subsecciones.

4.1.1 Algoritmo de verificación

Para el cálculo de rendimiento se ha creado una clase auxiliar que no depende de ninguna de las mostradas en el diagrama de clases. Dicha clase, calcula el número de etiquetas diferentes en dos ficheros en función de la medida F, basándose en el siguiente algoritmo. Fuente (***arnavk/NLP*** 2014).

```
public static RendimientoModelo validate(String outputFile, String
inputFile) {

List<List<String>> outputList = getTagListForFile(outputFile);
List<List<String>> correctedList = getTagListForFile(inputFile);

/* Contadores de datos */
int tp = 0; /* True positives */
int fp = 0; /* False positives */
int fn = 0; /* False negatives */
int ttInput = 0; /* Total de etiquetas en el fichero de entrada */
int ttOutput = 0; /* Total de etiquetas en el fichero de salida */

for (int i = 0; i < outputList.size(); i++) {
      tp--; /* Quitar la etiqueta ROOT, siempre va a ser correcta */
      List<String> outputTreeList = outputList.get(i);
      List<String> correctedTreeList = correctedList.get(i);

      ttInput += correctedTreeList.size() - 1;
      ttOutput += outputTreeList.size() - 1;

      int fnForTree = correctedTreeList.size();
      for (String node : outputTreeList) {
            if (correctedTreeList.contains(node)) {
                  tp++;
                  fnForTree--;
            } else {
                  fp++;
            }
      }
      fn += fnForTree;
}
```

```
return new RendimientoModelo(tp, fp, fn, ttTrain, ttOutput);
}
```

El primer paso del algoritmo es leer las etiquetas de los ficheros a comparar. Una vez obtenidas las etiquetas se pasa a obtener los diferentes datos: *true positives*, *false positives* y *false negatives*[5]. El algoritmo obtiene cada árbol a comparar en forma de lista de etiquetas tras ello, campara una a una las etiquetas. Si las etiquetas coinciden en la misma posición, estamos ante un *true positive*. En cambio, si en la lista de etiquetas del fichero de salida se encuentra una etiqueta que no está en la misma posición que en la lista de entrada, esto será tratado como un *false positive*. El cálculo de *false negatives* se realiza restando al número total de etiquetas en la lista de entrada a las etiquetas correctas, pues el número de etiquetas no analizadas serán etiquetas que no se encuentran el fichero de salida.

Una vez que el algoritmo ha terminado, los datos son almacenados en la siguiente clase:

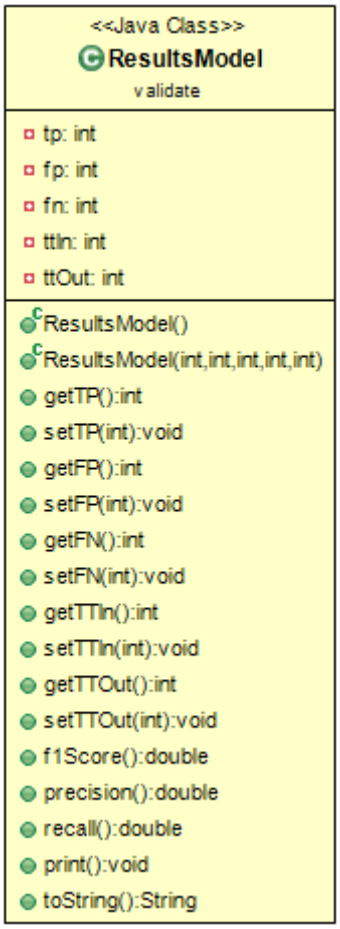


Figura 9. **Clase que almacena los datos obtenidos de la evaluación de rendimiento en función de la media F.**

Calculando los datos con una clase auxiliar y almacenando el resultado en otra clase, se proporciona flexibilidad ante un cambio de medida a la hora de obtener el rendimiento.

[5] Para mas información sobre estos datos véase 3.1.4

De este modo, si en un futuro se desease aplicar otro tipo de medida u otro algoritmo para el cálculo de la misma es posible una fácil adaptación.

4.1.2 Algoritmo de creación de árboles

El algoritmo de creación de árboles está adaptado al *treebank* empleado para el desarrollo del proyecto, ajustándose a las etiquetas que éste contiene y las estructuras que utiliza. Por ello es importante destacar que este algoritmo no es general para cubrir cualquier tipo de *corpus* utilizado. Como se puede comprobar, para las diferentes herramientas se han empleado dos algoritmos diferentes ajustándose a las necesidades de cada una de ellas.

En este apartado nos centraremos en el algoritmo utilizado para leer los árboles dentro de la herramienta de validación.

```
public Arbol createTree(String arbol) {
	Arbol tree = new Arbol();
	ArrayList<Nodo> nodos = new ArrayList<Nodo>();
	Stack<Nodo> stack = new Stack<Nodo>();
	String valor = new String();
	Boolean esValor = false, meterValor = false;

	String[] s = arbol.split("\\s");
	for (int i = 0; i < s.length; i++) {
		/* Para evitar iteraciones vacias */
		if (s[i].equals("")) {
			continue;
		}

		/* Comienzo de nodo */
		if (s[i].startsWith("(")) {
			/* Crear nodo y guardar clave */
			Nodo nodo = new Nodo();
			String clave = s[i].replace("(", "");
			clave = clave.replace(")", "");
			nodo.setClave(clave);
			/* Incluir a la lista de nodos */
			nodos.add(nodo);

			/* Guardar los hijos */
			if (!stack.isEmpty()) {
				/* Nodos hijos del arbol */
				Nodo nodoPadre = stack.pop();
				nodoPadre.addHijo(nodo);
				nodo.setPadre(nodoPadre);
				stack.push(nodoPadre);
			}

			/* Meter a la pila */
			stack.push(nodo);
		}

		if (s[i].startsWith("\"")) {
			/* Valor para asignar */
			valor = s[i].replaceFirst("\"", "");
			esValor = false;
		}
```

```
                /* Si el valor no ha terminado de leerse */
                if (esValor) {
                      valor += " " + s[i];
                }

                /* Si termina el valor */
                if (valor.contains("\"")) {
                      valor = valor.replace(")", "");
                      valor = valor.replace("\"", "");

                      /* Si el valor es un ')' */
                      if (valor.contentEquals("/PUNCT"))
                            valor = ")" + valor;

                      meterValor = true;
                      esValor = false;
                } else {
                      esValor = true;
                }

                /* Meter el valor calculado anteriormente */
                if (meterValor) {
                      Nodo nodoActual = stack.pop();
                      nodoActual.setValor(valor);
                      stack.push(nodoActual);
                      meterValor = false;
                      valor = new String();
                }

                /* Terminar nodo (pueden ser varios a la vez) sacar de la
pila */
                if (s[i].endsWith(")")) {
                      int idx = s[i].lastIndexOf(")");
                      int j = 0;
                      while (s[i].charAt(idx - j) == ')') {
                            j++;
                            stack.pop();
                      }
                }
}
```

El algoritmo está desarrollado para leer un árbol desde un fichero en texto plano que respete las mismas etiquetas y estructuras del texto. Para hacer posible la lectura de los árboles a memoria, se va leyendo línea a línea el fichero. Una vez leída una línea primero se procede a separar en *tokens*[6] o palabras para analizarlas una a una.

Para gestionar los nodos del árbol se emplea una lista de nodos, así como una pila para mantener la concordancia entre padres e hijos. Además la pila es un útil para, una vez termina la lectura de una árbol, comprobar si está bien balanceado o no.

[6] *Token* se refiere a cualquier carácter posible como ".", ")", etc., o conjunto de los mismo "\.", "\(", etc., así como al conjunto de una etiqueta con un carácter, a una etiqueta sola, o a cualquier otro elemento separado que pueda aparecer en la lectura del árbol.

Dependiendo del tipo de nodo, padre o hijo, y el tipo de *token* leído, etiqueta, carácter de cierre, etc., el algoritmo realiza una serie de acciones[7] posibles:

- Si el *token* leído es un " (" (paréntesis de apertura), se crea un nuevo nodo y se incluye a la lista de nodos del árbol, así como a la pila. También se comprueba si es padre de modo que si existe un nodo dentro de la pila, entonces ese es padre del nuevo nodo. Se asigna la categoría al nodo eliminando los paréntesis, tanto de cierre como de apertura.

- Si el *token* comienza con " " " (Comillas dobles), significa que se empieza a leer la palabra o palabras del nodo.

- Si el *token* termina con " " " (Comillas dobles), entonces se termina de leer la palabra o palabras del nodo. En este caso hay que tener cuidado si se trata de un *token* de cierre de paréntesis, pues esta eliminado con anterioridad y hay que incluirlo manualmente.

- Si el *token* leído termina con ") " (paréntesis de cierre), significa que es el nodo termina, y hay que extraerlo de la pila, pues ya no puede ser padre de ningún otro modo. En este caso como es posible que aparezcan varias veces seguidas, se extraen tantos nodos de la pila como numero de apariciones tenga el paréntesis de cierre.

Una vez leído el árbol se incluye un nodo raíz cuya etiqueta es *ROOT* de la cual cuelga todo el árbol. Esta etiqueta es necesaria dado que el software del Stanford Parser siempre la incorpora.

4.2 Software de generación

El software de generación crea, de forma automática, nuevas frases con las que incrementar el *treebank* original. El software se ajusta al siguiente diagrama de clases:

[7] En el algoritmo de la herramienta de generación aparecen más acciones, como leer el número lingüístico que tiene asociado cada sintagma.

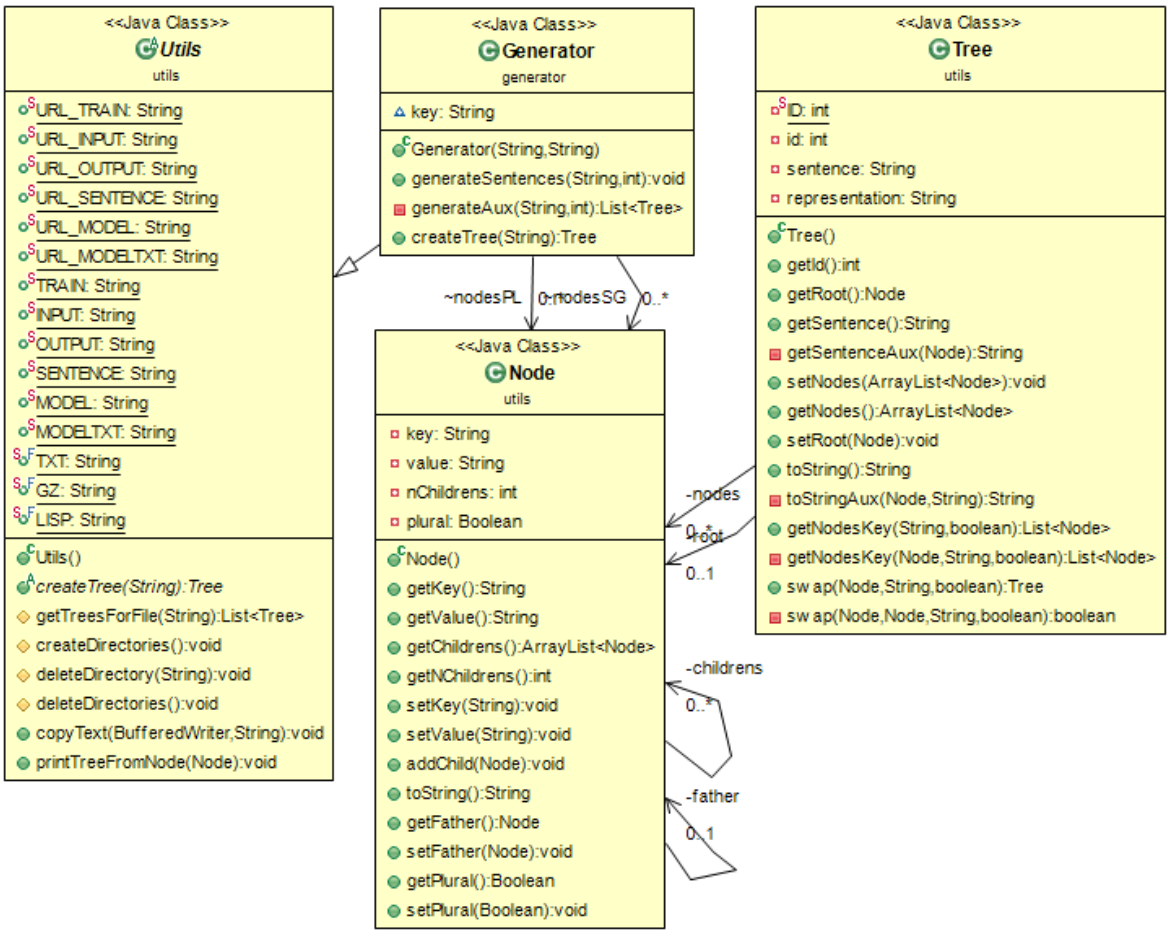


***Figura 10*. Diagrama de clases de la herramienta de generación.**

Para el funcionamiento de esta herramienta son necesarios, al menos, dos ficheros externos, uno del cual extraer los sintagmas para realizar el intercambio, y otro que guarda las estructuras de las cuales crear nuevas frases.

5 Integración, pruebas y resultados

En este apartado se describen las distintas pruebas que se han llevado a cabo para certificar el uso correcto de las herramientas. Para ello se distinguen dos tipos, pruebas unitarias, donde se comprueba cada clase por separado, y pruebas de integración donde se comprueba el funcionamiento global de las herramientas.

5.1 Pruebas unitarias

En la realización de las pruebas unitarias, se distinguen dos pruebas, las pruebas de caja blanca las cuales se centran en el comportamiento interno lógico. Y las pruebas de caja negra en las cuales los datos de entrada deben coincidir con la salida esperada.

5.1.1 Pruebas de caja blanca.

Estas pruebas consisten en probar los caminos básicos de los algoritmos empleados en las diferentes herramientas, para ello partiendo del código:

- Se dibuja el grafo de flujo
- Se calcula la complejidad ciclomática.
- Se determinan los caminos linealmente independientes.
- Se preparan los casos de prueba que forzarán la ejecución en cada camino.

A modo de ejemplo se describe la prueba realizada a la función *getTreesForFile* de la clase *Utils*.

```
protected List<Arbol> getTreesForFile(String filename) {
    BufferedReader file;
    List<Arbol> trees = new ArrayList<Arbol>();
    String s;
    String tree = new String();

    try {
        file = new BufferedReader(new FileReader(filename));

        while ((s = file.readLine()) != null) {          // A

            if (s.equals("") && !tree.isEmpty()) {       // B
                Arbol arbol = createTree(tree);          // C
                if (arbol != null)
                    trees.add(arbol);
                tree = new String();
            } else if (!s.equals("")) {                  // D
                tree += s;
            }                                            // E
        }

    } catch (IOException e) {
        e.printStackTrace();
    }

    return trees;                                        // F
}
```

***Figura 11*. Código.**

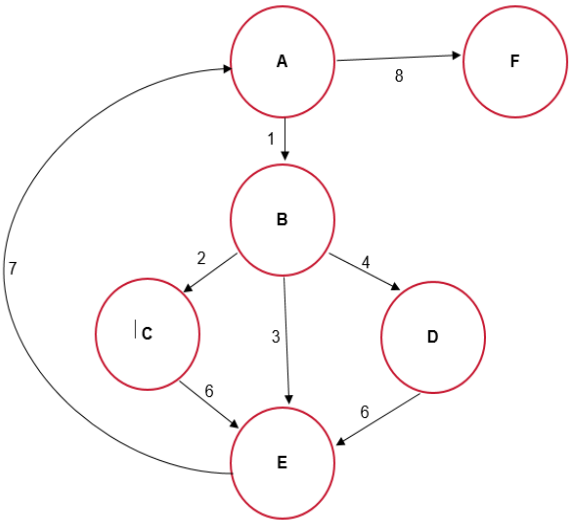


***Figura 12*. Grafo de flujo.**

Para calcular la complejidad ciclomática nos ajustamos a la siguiente igualdad, siendo G el grafo de caminos:

$$V(G) = Nº\ Aristas - Nº\ nodos + 2$$

En este caso

$$V(G) = 8 - 6 + 2 = 4$$

	1	2	3	4	5	6	7	8	Casos de prueba
AF	0	0	0	0	0	0	0	1	(s = file.readLine()) == **null**
ABCEAF	1	1	0	0	1	0	1	1	(s = file.readLine()) != **null**, s.equals(""), !tree.isEmpty()
ABDEAF	1	1	0	1	0	1	1	1	(s = file.readLine()) != **null,** !s.equals("")
ABEAF	1	0	1	0	0	0	1	1	(s = file.readLine()) != **null**

***Tabla 1*. Casos de prueba de los caminos.**

Se han revisado las funciones con más de 20 líneas, ninguna de ellas tiene una complejidad mayor a 10. Tan solo hay dos funciones que podrían tener un riesgo moderado, pues tienen una complejidad de 10. El resto de funciones son fáciles de mantener y de entender.

5.1.2 Pruebas de caja negra

Para estas pruebas se hace uso del *plugin* de eclipse, *JUnit*, con las cuales se prueban las diferentes clases una por una de manera específica. La tabla siguiente muestra un resumen de las pruebas realizadas:

CLASE	MÉTODO	DESCRIPCIÓN
TREE	Sentence	Prueba la extracción de la oración desde el árbol que la contiene.
	Root	Prueba que la raíz del árbol sea la correcta.
	Tree	Comprueba que el árbol ha sido creado correctamente.
	nodeKey	Comprueba que se obtienen los nodos con la clave elegida.
	Swap	Comprueba que la sustitución de nodos se realiza correctamente.
UTILS	deleteDirectories	Comprueba que se los directorios temporales son borrados correctamente.
	createDirectories	Comprueba que los directorios temporales son creados correctamente.
	getTreeForFile	Comprueba que los árboles son leídos correctamente en memoria desde un archivo.
	printTreeForNode	Comprueba que a partir de un nodo se crea su representación del árbol tomando el nodo dado como raíz.

VALIDATOR	calculateIndex	Comprueba que los índices son calculados correctamente.
	createTree	Comprueba que dada una cadena que contiene un árbol, este es almacenado en memoria en forma de árbol.
SIMPLEVALIDATOR	createFiles	Comprueba que los archivos necesarios para la validación simple son creados correctamente.
	Validate	Comprueba que la validación simple es ejecutada correctamente.
	externalValidate	Comprueba que la validación externa es ejecutada correctamente.
CROSSVALIDATOR	createFiles	Comprueba que los archivos necesarios para la validación cruzada son creados correctamente.
	Validate	Comprueba que la validación cruzada es ejecutada correctamente.
TREEVERIFICATOR	Validate	Comprueba que la comparación de etiquetas es ejecutada correctamente.
GENERATOR	generateSentences	Comprueba que el número y las sentencias generadas son correctas.
	createTree	Comprueba que dada una cadena que contiene un árbol, este es almacenado en memoria en forma de árbol[8].

***Tabla 1*. Resumen de las pruebas de caja negra realizadas.**

5.2 Pruebas de integración

Las pruebas de integración están diseñadas para comprobar el correcto funcionamiento de las herramientas, para ello, se han diseñado una serie de pruebas específicas para cada herramienta, donde se verifica el funcionamiento de cada una de ellas.

5.2.1 Validación.

En este conjunto de pruebas se han probado las funcionalidades básicas de la herramienta, existen multitud de combinaciones posibles de parámetros (*A.A*) y por ello no se han probado todas[9].

Parámetros	Salida esperada
	Introducir al menos los parámetros "-tb" "url fichero treebank"

[8] Misma prueba que para la clase *validator*, hay que tener en cuenta que los algoritmos sufren unas pequeñas modificaciones, en concreto el algoritmo de la clase *generator* tiene más acciones a realizar (véase 4.1.2)

[9] Los ficheros son citados como “fichero x”, en este caso debería aparecer la ruta hasta el fichero.

-tb “fichero treebank”	Validación interna simple
-tb “fichero treebank” -c	Validación interna cruzada con 10 particiones.
-tb “fichero treebank” -c -n 20	Validación interna cruzada con 20 particiones.
-tb “fichero treebank” -size 400 –s	Validación interna simple con un tamaño de conjunto de 400 elementos.
-tb “fichero treebank”-e -input “fichero input” -sentences “fichero sentences”	Validación externa
-i –e	Incompatibilidad en las opciones -i -e no son compatibles
-size 10	Falta el treebank de entramiento
-tb “fichero treebank” -e -input “fichero input”	Faltan las frases para validar
-tb “fichero treebank” -e -sentences “fichero sentences”	Faltan las frases bien etiquetadas para calcular rendimiento

***Tabla 2*. Conjunto de pruebas de integración de la herramienta de validación.**

En este conjunto se tienen en cuenta tanto todas las opciones de ejecución como los errores que se pueden llegar a producir. Estos los test fueron superados por la herramienta.

5.2.2 Generación de frases

Para probar la herramienta de generación de frases, se escogió una estructura de oración y a partir de ésta se crearon 10 nuevas frases. Tras crear las frases se revisaron manualmente para comprobar que las frases artificialmente creadas eran coherentes. A continuación, se muestra un ejemplo de la prueba a partir de la frase original “*Vive el mono en Asia.*”:

Vive/V La/ART cumbre/N monográfica/ADJ de/PREP Luxemburgo/N sobre/PREP empleo/N ./PUNCT
Vive/V Yeltsin/N ./PUNCT
Vive/V las/ART normas/N ./PUNCT
Vive/V el/ART premio/N de/PREP los/ART editores/N de/PREP Madrid/N ./PUNCT
Vive/V la/ART gana/N ./PUNCT
Vive/V la/ART ofensiva/N contra/PREP el/ART ELK/N ./PUNCT
Vive/V el/ART juego/N a/PREP Bolsa/N ./PUNCT Vive/V todos/Q los/ART rincones/N de/PREP la/ART península/N ./PUNCT
Vive/V la/ART versión/N de/PREP los/ART peritos/N en/PREP Carburos/N ./PUNCT
Vive/V la/ART envidiable/ADJ ciudad/N de/PREP Río de Janeiro/N ./PUNC

6 Experimentación y resultados

La hipótesis de partida que se maneja es que con un mayor entrenamiento, se obtendrán mejores resultados a la hora de etiquetar frases. Para verificar la hipótesis de partida, se han realizado varias pruebas[10], partiendo de un *treebank* con 1600 frases.

6.1 Analizar la capacidad de aprendizaje del Stanford Parser: primera prueba

La primera prueba consistió en analizar la capacidad de aprendizaje del analizador sintáctico con el *treebank* usado. Para ello se realizaron dos pruebas en paralelo en las cual se obtuvo la curva de aprendizaje analizada desde la vista de la validación cruzada, y desde la perspectiva de la validación simple, es decir, utilizando el mismo conjunto para entrenar y para realizar el test.

6.1.1 Validación interna

La prueba interna consiste en, partiendo del *treebank* original y utilizando solo los datos contenidos en el mismo, hacer una validación del analizador sintáctico para obtener el rendimiento del mismo. Esta prueba se bifurca en dos ramas, ambas mantienen el mismo conjunto de frases, de modo que se puede comparar de manera fiable los resultados obtenidos.

La primera entrena un número "N" de oraciones y realiza el test de las mismas. Los resultados obtenidos son los siguientes:

N	*F1Score*	*Precision*	*Recall*
10	0.2249	0.2191	0.2310
20	0.2922	0.2843	0.3006
50	0.3710	0.3631	0.3793
80	0.3673	0.3577	0.3774
100	0.3730	0.3631	0.3835
200	0.4006	0.3900	0.4117
400	0.3733	0.3613	0.3862
800	0.3818	0.3711	0.3931
1600	0.3730	0.3619	0.3845

***Tabla 3*. Resultados del estudio de aprendizaje del Stanford Parser usando validación simple.**

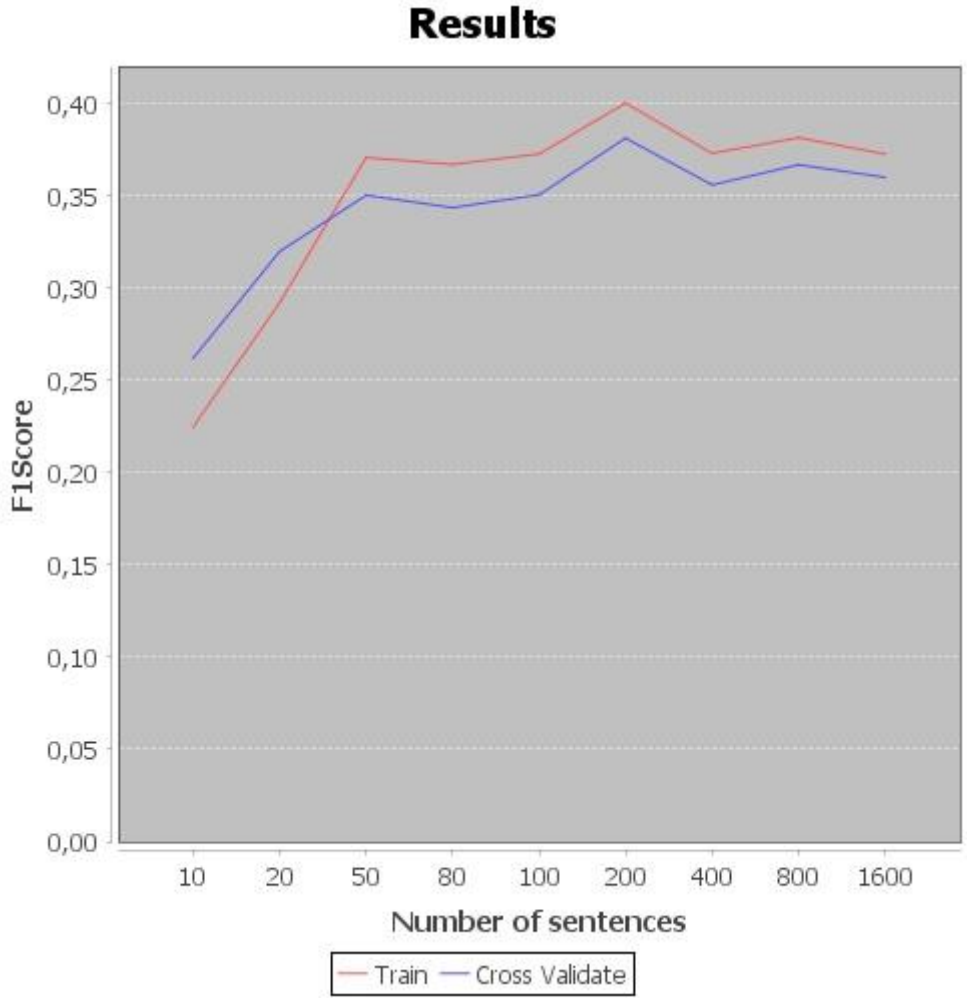
[10] Para mantener la concordancia y facilitar el entendimiento, los resultados están redondeados a las diez milésimas.

La segunda, entrena con un tamaño "N" del conjunto de frases y realiza la validación cruzada escogiendo "M" particiones. En este caso "M" es fijo en 10, lo que se traduce en un entrenamiento con 9 particiones y 1 solo conjunto para validar.

N	*F1Score*	*Precision*	*Recall*
10	0.2621	0.2572	0.2673
20	0.3201	0.3129	0.3276
50	0.3506	0.3422	0.3594
80	0.3439	0.3337	0.3546
100	0.3509	0.3398	0.3626
200	0.3815	0.3701	0.3938
400	0.3562	0.3412	0.3725
800	0.3671	0.3542	0.3810
1600	0.3603	0.3481	0.3734

***Tabla 4*. Resultados del estudio de aprendizaje del Stanford Parser usando validación cruzada.**

Con los datos obtenidos en las pruebas, realizamos una gráfica para mostrar las curvas de aprendizaje. En estas curvas se puede ver que ambas tienen un recorrido similar. Pues como es de esperar, al emplear el mismo conjunto los resultados han de ser parecidos.



***Gráfica 1*. Curvas de aprendizaje obtenidas en la prueba para analizar la capacidad de aprendizaje del Stanford Parser empleando datos de validación internos.**

Podemos ver qué en una primera aproximación, la hipótesis enunciada no iba mal encaminada, pues las curvas de aprendizaje basadas en el número de frases y la puntuación obtenida, evolucionan de un modo positivo.

Los picos que se producen son debido al factor de aleatoriedad que está siempre presente. Al leer los árboles en memoria se desordenan todos para asegurarnos de que la colocación no influye en los resultados. El factor produce que en ocasiones las oraciones se etiqueten mejor, debido al entrenamiento que ha tenido el analizador sintáctico (*A.C*).

6.1.2 Validación externa

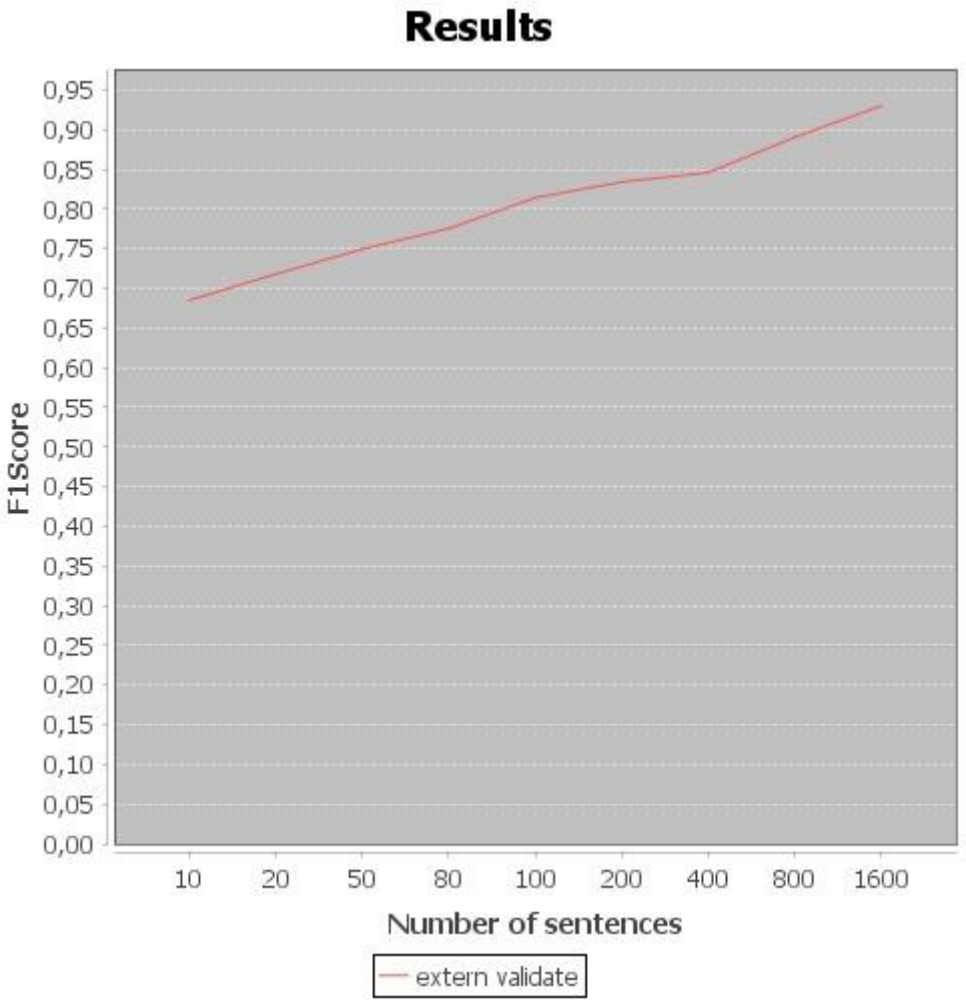
En esta ocasión, las frases que se validan serán datos externos, para ello se irá aumentando el tamaño del conjunto de entrenamiento escogido para ver si, efectivamente, un mayor entrenamiento se traduce en mejores resultados.

En esta prueba se realiza una validación simple, en la cual se entrena con un conjunto *N* de datos y se valida con un el mismo conjunto externo.

El conjunto externo contiene un total de 18 frases (*A.D*) escogidas por el departamento de lingüística y del cual se han proporcionado las salidas correctas para poder proceder a la obtención y análisis del rendimiento. Este conjunto abarca las estructuras más comunes del español, para, de este modo, poder analizar donde se comenten más fallos e intentar resolverlo en un futuro.

N	*F1Score*	*Precision*	*Recall*
10	0.6848	0.6700	0.7003
20	0.7179	0.6948	0.7427
50	0.7493	0.7378	0.7613
80	0.7753	0.7682	0.7825
100	0.8141	0.8036	0.8249
200	0.8340	0.8222	0.8461
400	0.8458	0.8403	0.8514
800	0.8903	0.8868	0.8939
1600	0.9298	0.9285	0.9298

***Tabla 5*. Resultados del estudio de aprendizaje del Stanford Parser usando datos externos.**



Gráfica 2. **Curvas de aprendizaje obtenidas en la prueba para analizar la capacidad de aprendizaje del Stanford Parser empleando datos de validación externos.**

Los datos en este caso reflejan claramente la suposición de la que partimos, pues al aumentar el número de frases para entrenar, cada vez se va mejorando el resultado hasta la abismante puntuación de 0,9298 de *F1Score*. Recordemos que en la puntuación F1 cuanto más se acerca a 1 mejor es el resultado, lo que en nuestro caso significa un resultado casi perfecto.

En este punto lo que se intenta es afinar el resultado hasta alcanzar el valor más próximo a una puntuación perfecta. Para ello interviene el factor humano, analizando de forma manual las frases en las que se comete error[11].

En este análisis descubrimos un patrón común derivado de las estructuras similares de algunas frases. Estas oraciones se definen de modo que el predicado, y por lo tanto el verbo, se encuentra antes del sujeto. Esto hace cometer errores al etiquetador del analizador sintáctico dando a entender que un sintagma nominal precedido de un verbo debe ser un

[11] El análisis de datos manualmente es posible debido a que el número de frases que empleamos no es muy elevado, y que los fallos cometidos son muy pocos, con un conjunto más grande y mayor cantidad de fallos este análisis se haría muy tedioso.

objeto, ya sea directo o indirecto, en lugar de un sujeto. A continuación se muestran una serie de frases donde se aprecia dicho patrón:

```
(ROOT
 (S
  (VP (V Informó))
  (NPSUBJ (ART el) (N presidente))
  (CL (C que)
     (VP
      (ADVP (ADV no))
      (V habrá)
      (NPOBJ1 (N aumentos)
              (PP (PREP de)
                    (NP (N sueldo))))))
  (PUNCT .)))
```

```
(ROOT
 (S
  (VP
   (VP (V Informó)
    (NPOBJ1 (ART el) (N presidente)))
   (C que)
   (VP
    (ADVP (ADV no))
    (V habrá)
    (NPOBJ1 (N aumentos)
     (PP (PREP de)
      (NP (N sueldo))))))
  (PUNCT .)))
```

```
(ROOT
 (S
  (VP (V Muere)
     (PP (PREP en)
          (NP (NPR París))))
     (NPSUBJ (ART el) (N maestro)
             (PP (PREP de)
                   (NP (N actores)))
  (NP (NPR Jacques)))
  (PUNCT .)))
```

```
(ROOT
 (S
  (VP (V Muere)
   (PP (PREP en)
    (NP (NPR París)))
   (NPOBJ1 (ART el) (N maestro)
    (PP (PREP de)
     (NP (N actores)))))
  (NPSUBJ (NPR Jacques))
  (PUNCT .)))
```

```
(ROOT
 (S
  (VP (V Muere)
   (NPOBJ1 (ART el) (N presidente)
    (PP (PREP de)
     (NP (N Francia)))))
  (NPSUBJ (NPR Pepe))
  (PUNCT .)))
```

```
(ROOT
 (S
  (VP (V Muere))
     (NPSUBJ (ART el) (N presidente)
             (PP (PREP de)
                   (NP (N Francia))
  (NP (NPR Pepe))))
  (PUNCT .)))
```

6.1.3 Análisis de los errores más frecuentes

Con una buena impresión de que la hipótesis en la que estamos trabajando es correcta, se realizó otra prueba en la cual se obtienen las frases con puntuaciones de medida F mayores o iguales a las cotas de 0.5, 0.6, 0.7, 0.8, 0.9 y 1.

Con estas frases lo que se quiere es revisar por una parte, que el algoritmo empleado para la obtención del rendimiento es fiable y por tanto valido. Y por otro lado, se quiere ver cuantas frases, y cuales, de las 1600 que componen el *treebank* se etiquetan con mejor puntuación, para un posible estudio de patrones entre estas frases.

Para realizar la prueba se entrenó el analizador sintáctico con 1599 frases y se realizó la validación con la frase restante. A continuación, se muestran los datos obtenidos. La columna de la izquierda muestra la puntuación y la columna de la derecha muestra el número de frases que obtuvieron dicho resultado:

F1SCORE[12] [13]	Nº SENTENCES
>= 0.5	604
>= 0.6	524
>= 0.7	463
>= 0.8	255
>= 0.9	16
== 1	0

***Tabla 6*. Numero de frases con rendimiento mayor o igual a una puntuación.**

Es importante tener en cuenta que el resultado es acumulativo, es decir, si una frase obtiene una puntuación perfecta, de 1, entonces esta frase aparecerá en todos los resultados, pues cumple que sea mayor o igual a 0.5, a 0.6, a 0.7, a 0.8, a 0.9 y que es igual a 1.

Los resultados son muy bajos, ni una sola de las frases se etiqueta correctamente al 100%, tan solo el 1% de las frases se etiqueta con una puntuación mayor o igual a 0.9.

El corte de puntuación aceptada suele estar en torno al 0.7, 0.8 y podemos ver que el porcentaje de frases obtenidos en estos casos es de 28,9% y 15,9% respectivamente. Así que, la conclusión de esta prueba es que el analizador sintáctico obtiene un mal rendimiento, pues ni si quiera un 30% de las frases se etiqueta con una puntuación del 0.7.

En esta prueba se compararon a mano las frases y se obtuvo un resultado muy interesante pues se apreció que las etiquetas tienden a generalizarse de tal forma que las etiquetas compuestas, es decir las etiquetas separadas por el símbolo "_" , tan solo leen la primera parte de la etiqueta. Por ejemplo, si tenemos una etiqueta VP_TENSED y otra etiqueta VP_UNTENSED estas dos serán la misma y se reducirán a VP. Esto provoca por una parte que no exista la diferencia de etiquetas que comienzan con el mismo valor, por lo tanto, la perdida de información. Y por otro lado esto genera errores en la comparación del rendimiento, pues si estamos comparando una etiqueta compuesta, por ejemplo, VP_TENSED y el resultado es tan solo VP, la comparación generara un error, lo que se traduce en un peor rendimiento (*A.E*).

Para subsanar los errores provocados se modificó el *treebank* empleado, modificando las etiquetas que seguían el patrón de fallo, VP_TENSED y VP_UNTENSED, convirtiéndolas en VPTENSED y VPUNTENSED. Con este nuevo *treebank* se realizó la misma prueba y se obtuvieron los siguientes resultados:

F1SCORE	Nº SENTENCES
>= 0.5	615
>= 0.6	533

[12] >= se refiere a la puntuación mayor o igual obtenido al valor que lo sigue.

[13] == puntuación solo igual al valor que lo sigue

>= 0.7	482
>= 0.8	368
>= 0.9	88
== 1	13

***Tabla 7*. Numero de frases con rendimiento mayor o igual a una puntuación empleando un treebank con dos nuevas etiquetas VPTENSED y VPUNTENSED.**

También se quiso estudiar el efecto de eliminar las etiquetas convirtiéndolas en VP directamente, para ello se realizó la misma prueba y se obtuvieron los siguientes resultados.

F1SCORE	Nº SENTENCES
>= 0.5	615
>= 0.6	531
>= 0.7	482
>= 0.8	365
>= 0.9	87
== 1	13

***Tabla 8*. Numero de frases con rendimiento mayor o igual a una puntuación empleando treebank con etiquetas generalizadas a VP.**

Como se muestra los resultados de ambas pruebas son muy parecidos, pero en la primera donde se diferencian dos etiquetas para el verbo, el resultado es mejor. Esto es muy interesante pues se consigue introducir dos nuevas etiquetas mejorando el rendimiento y no empeorándolo como se creía al principio.

Cabe destacar que con estas nuevas etiquetas se consigue que 13 frases, el 0.8%, se etiqueten perfectamente, de modo que haga que incremente el rendimiento del SP. También se consigue pasar del 1% de frase con una puntuación del 0.9 a un 5.5% más de un 4% de incremento. Pero lo más extraordinario es que con una puntuación mayor a 0.8 hemos conseguido un incremento del 6% pasando del 15.9% a un 22.8%.

En general estos resultados nos dicen que más frases consigan mejor puntuación se refleja en un etiquetado más preciso de otras nuevas frases.

6.2 Mejorando la validación externa: entrenando al verbo

Ante los resultados obtenidos en la prueba anterior y habiendo avanzado en la hipótesis formulada al principio, procedemos a realizar una prueba en la cual, partiendo de una serie de estructuras, se combinarán y crearán nuevas oraciones de estructura similar para intentar enseñar al analizador sintáctico la función de los sintagmas nominales dependientes del verbo.

Para ello se cogen tres estructuras de oraciones similares a las que en la primera prueba se averiguo que más fallos provocaban, de modo que, intercambiando los sintagmas nominales que desempeñan la función de sujeto por otro sintagma nominal el cual realizará la misma función, generamos nuevas frases con las que aumentaremos el número de frases del *treebank*.

En la prueba se fueron añadiendo frases por cada iteración para poder estudiar la evolución de la validación y ver cómo afecta la inclusión de nuevas frases, no solo a las frases mal etiquetadas, si no a todas las frases, pues el objetivo es arreglar los fallos sin incurrir en uno nuevo.

Para hacer un reparto equitativo de las nuevas frases que se incluyen se decidió usar múltiplos de 3, ya que es el numero elegido de estructuras de las cuales se crearan nuevas frases. A continuación, se muestra una tabla en la cual se refleja el número de frases creadas y la puntuación que obtiene este nuevo *treebank* con validación externa.

NÚMERO DE FRASES NUEVAS	NÚMERO DE FRASES EN EL TREEBANK	F-SCORE
3	1603	0.9298
6	1606	0.9298
9	1609	0.9298
12	1612	0.9298
15	1615	0.9298
18	1618	0.9298
21	1621	0.9298
24	1624	0.9298
27	1627	0.9298
30	1630	0.9298
60	1660	0.9324
90	1690	0.9377
120	1720	0.9377
240	1840	0.9285
360	1960	0.9285
480	2080	0.9273
600	2200	0.9273
900	2500	0.9155
1200	2800	0.8812

***Tabla 9*. Resultados de la validación externa con un *treebank* extendido con datos artificiales**

Como se puede comprobar se consigue mejorar el rendimiento obtenido por el Stanford Parser, empezamos con un rendimiento de 0.9298, generando nuevas frases y entrenando el Stanford Parser con un conjunto de frases que incluye las nuevas frases más las antiguas, se consigue obtener un rendimiento de 0.9377.

Por otro lado se puede observar que un sobre entrenamiento es tan nocivo como es entrenar poco el clasificador, pues puede hacer que aprenda mal el clasificador y obtenga una puntuación peor que la obtenida con menos frases.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

La adaptación del Stanford Parser al español no depende de una sola persona, sino de una comunidad que trabaja sin ánimo de lucro y con un único fin de obtener un software competente dentro del campo de los analizadores sintácticos computacionales. Al ser una comunidad la encargada del desarrollo del proyecto se descartó la primera línea de trabajo ya que por tiempo no se podía abordar este trabajo de manera individual.

Centrándonos en la segunda línea de trabajo, y entendiendo como funciona el analizador sintáctico es posible acercarse a un mejor rendimiento, por ejemplo, creando las etiquetas justas del *treebank* y afinándolas en la medida de lo posible.

En cuanto a las herramientas propuestas al principio del trabajo, se ha conseguido el objetivo de crear una herramienta de validación, que funciona correctamente y proporciona el rendimiento del Stanford Parser.

Por último, la herramienta de generación de frases ha sido desarrollada con solo una estrategia, la cual consiste en combinar los sintagmas nominales sujeto. Gracias a esto, se ha conseguido entrenar al verbo para que aprenda la función de sujeto cuando la estructura de la oración es verbo + sujeto, pues este era uno de los fallos descubiertos que más se cometía, el analizador lo trataba como verbo + objeto.

7.2 Trabajo futuro

Como trabajo futuro, se puede seguir trabajando en el proyecto del Stanford Parser aportando cualquier tipo de herramienta, que proporcione nuevas funcionalidades al analizador sintáctico o mejore las que ya posee. También, se puede trabajar directamente en el software para incorporar las anotaciones soportadas para el español, pues por aquí es por donde más rápido se pude avanzar y donde mejor resultado se obtendrá.

Otro punto donde trabajar y mejorar dicha adaptación es desarrollando aún más el *treebank* empleado. En esta línea se puede trabajar en conjunto con el Departamento de Lingüística Computacional de la Universidad Autónoma de Madrid para alcanzar el objetivo de 5000 frases. Logrando así, crear un *treebank* que contemple los rasgos más significativos del lenguaje humano español.

Por otro lado, las herramientas creadas para este Trabajo de Fin de Grado pueden explotarse mucho más. Concretamente, la herramienta de generación de frases se puede mejorar para dar cabida a cualquiera de las otras estrategias planteadas en la generación de frases, así como a nuevas estrategias que se planteen en un futuro.

La herramienta de validación también tiene un largo recorrido, se puede adaptar para emplear otros tipos de validación, sin olvidar que también se puede adaptar para emplear

otras medidas de la misma. Además, al ser una herramienta flexible que trabaja con ficheros externos se puede generalizar para proporcionar utilidad a otros proyectos similares y clasificadores.

Por último, a ambas herramientas se les puede incluir interfaz gráfica para hacerlas más atractivas e intuitivas.

Referencias

1. ALCINA, Amparo; VALERO, Esperanza. *Terminología y sociedad del conocimiento*. Peter Lang, 2009.

2. Arnavk/NLP, 2014. *GitHub* [En línea]. Disponible: https://github.com/arnavk/NLP [Último acceso: 2016].

3. CHOMSKY, Noam. Nuestro Conocimiento del Lenguaje Humano: Perspectivas Actuales. *Our Knowledge of Human Language: Current Perspectives*, 1998.

4. Definición de lingüística — Definicion.de, 2016. *Definición.de* [En línea]. Disponible: http://definicion.de/linguistica/ [Último acceso: 2016].

5. GUINOVART, Javier Gómez. Fundamentos de Lingüística Computacional: bases teóricas, líneas de investigación y aplicaciones. *Bibliodoc: anuari de biblioteconomia, documentació i informació*, 1998, p. 135-146.

6. Introducción a la Lingüística, 2016. *Aucel.com* [En línea]. Disponible: http://www.aucel.com/articulos/ [Último acceso: 2016].

7. Laboratorio de Lingüística Informática, 2016. *Lllf.uam.es* [En línea]. Disponible: http://www.lllf.uam.es/ESP/ [Último acceso: 2016].

8. MANNING, Christopher D., et al. The Stanford CoreNLP Natural Language Processing Toolkit. En *ACL (System Demonstrations)*. 2014. p. 55-60.

9. MORENO SANDOVAL, Antonio, et al. A Treebank of Spanish and its Application to Parsing. En *LREC*. 2000.

10. MORENO SANDOVAL, Antonio. *Lingüística computacional*. Teide, 1998.

11. MORENO SANDOVAL, Antonio; CAMPILLOS LLANOS, Leonardo. Combinación de estrategias léxicas y estadísticas para el reconocimiento automático de términos: su aplicación a un corpus de medicina. *LEA: Lingüística española actual*, 2015, vol. 37, no 2, p. 175-199.

12. MORENO SANDOVAL, Antonio; GARROTE SALAZAR, Marta. La anotación de la negación en un corpus escrito etiquetado sintácticamente. *Revista Iberoamericana de Lingüística: RIL*, 2013, no 8, p. 45-60.

13. MORENO SANDOVAL, Antonio; GUIRAO MIRAS, José María. Frecuencia y distintividad en el uso lingüístico: casos tomados de la lematización verbal de corpus de distintos registros. En *A survey of corpus-based research [Recurso electrónico]*. 2009. p. 195-210.

14. MORENO SANDOVAL, Antonio; LÓPEZ RUESGA, Susana.; SÁNCHEZ LEÓN, Fernando. Spanish Tree Bank: Specifications. Version 5. 30 April 1999. 1999.

15. PERIÑÁN PASCUAL, José Carlos. En defensa del procesamiento del lenguaje natural fundamentado en la lingüística teórica. *Onomázein: Revista de Lingüística, Filología y Traducción*, 2012, no 26, p. 13-48.

16. Real Academia Española, 2016. *Rae.es* [En línea]. Disponible: http://www.rae.es/ [Último acceso: 2016].

17. SEGINER, Yoav, et al. Learning syntactic structure. 2007.

18. SIDOROV, Grigori. Problemas actuales de lingüística computacional. *Revista digital universitaria, UNAM, México*, 2001, vol. 2, no 1.

19. Stanford CoreNLP – a suite of core NLP tools | Stanford CoreNLP, 2016. *Stanfordnlp.github.io* [En línea]. Disponible: http://stanfordnlp.github.io/CoreNLP/ [Último acceso: 2016].

20. The Stanford Natural Language Processing Group, 2016. *Nlp.stanford.edu* [En línea]. Disponible: http://nlp.stanford.edu/software/lex-parser.shtml [Último acceso: 2016].

21. ZHENG, Alice, *Evaluating Machine Learning Models A Beginner's Guide to Key Concepts and Pitfalls*. O'Reilly, 2015.

Glosario

Corpus	Base de datos compuesta por oraciones en forma de árbol. También conocido como *treebank* o *Penn Treebank.*
Parser	Analizador sintáctico.
Stanford Parser	Analizador sintáctico computacional desarrollado por la Universidad de Stanford.
True positives (tp)	Unidad de medida empleada para el cálculo de *precision* y *recall*.
False negatives (fn)	Unidad de medida empleada para el cálculo de *recall*.
False positives (fp)	Unidad de medida empleada para el cálculo de *precision*.
Precision	Relación de datos utilizada para calcular el *F1Score*.
Recall	Relación de datos utilizada para calcular el *F1Score*.
F1Score	Medida de resultado empleada para calcular el rendimiento del Stanford Parser.
Javadoc	Utilidad de Oracle para la generación de documentación de APIs en formato HTML a partir de código fuente *java.*
JUnit	*Puglin* de eclipse utilizado para realizar pruebas unitarias de clases *java*.
Plugin	una aplicación que se relaciona con otra para agregarle una función nueva y generalmente muy específica.
Parsing	Termino para referirse al análisis sintáctico en español.
.jar	Es un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje *java*.
Java	Lenguaje de programación de propósito general, concurrente y orientado a objetos.

Anexos

A Manual de instalación

Las diferentes herramientas no requieren de ninguna instalación, tan solo es necesario descargarse el proyecto empaquetado como *.jar* desde su correspondiente dirección y emplear los parámetros correctamente para sus correspondientes ejecuciones.

Herramienta de validación del Stanford Parser.

La herramienta de validación está disponible a través del siguiente enlace través del repositorio "https://github.com/BorjaC/ValidadorSP/tree/master", pulsar "Clone or download" y a continuación "Download ZIP". Una vez descargado el archivo comprimido, descomprimir el archivo y obtener "validador.jar" empleado para la ejecución de la herramienta.

Para la ejecución es necesario:

- *Treebank* correctamente anotado.

Además para la validación de datos externos es necesario:

- Fichero *input* con la salidad de las oraciones que se quieren validar correctamente etiquetada.

- Fichero *sentences* con las frases que se quieren validar.

Los parámetros para la ejecución y sus acciones son los siguientes:

- **-tb "fichero treebank":** marca que lo siguiente leído será el fichero treebank utilizado en la validación. El "fichero treebank" hace referencia a la dirección completa de la ubicación y nombre del fichero.
- **-i:** indica que se va a realizar una validación interna de datos.
- **-e:** indica que se va a realizar una validación externa de datos.
- **-s:** indica que se va a realizar una validación simple de datos.
- **-c:** indica que se va a realizar una validación cruzada de datos, por defecto con 10 particiones.
- **-n "número entero":** indica el número de particiones usadas para la validación cruzada, "numero entero" será el número de particiones.
- **-size "número entero":** tamaño del conjunto para entrenar el Stanford Parser. Por defecto se usa el tamaño máximo del conjunto.
- **-input "fichero input":** marca que lo siguiente leído será el fichero *input* que contiene la salida de las frases que se van a validar correctamente etiquetadas.
- **-sentences "fichero sentences":** marca que lo siguiente leído será el fichero *sentences* que contiene las frases que se quieren validar.

Los parámetros por defecto son una validación interna simple con el tamaño del conjunto máximo. El orden de los parámetros se puede realizar de cualquier forma. A tener en cuenta:

- Los parámetros "**-i**" y "**-e**" son incompatibles.
- El tamaño del conjunto pasado no debe superar al número de elementos contenidos en el fichero. Si se quiere usar el máximo de elementos no poner tamaño.
- Validación interna simple se refiere a entrenar con todos los datos y validar todos.
- Validación interna cruzada se refiere a entrenar con n-1 conjuntos y validar con el conjunto restante.
- Validación externa se refiere a entrenar con el conjunto de datos del treebank y validar las frases del fichero *sentences*.
- Los parámetros -s -c son compatibles, los resultados aparecerán siempre en el orden de la validación interna simple primero y la validación interna cruzada después.

Ejemplos de ejecución:

java -jar validador.jar -size 400 tb "fichero treebank"

La salida obtenida será una validación interna con el tamaño de conjunto de 400 elementos.

java -jar validador.jar -tb "fichero treebank" -c

La salida obtenida será una validación interna cruzada usando todos los elementos del conjunto.

java -jar validador.jar -tb "fichero treebank" -input "fichero input" -sentences "fichero sentences" -e

La salida obtenida será una validación externa usando todos los elementos para entrenar Stanford Parser.

Herramienta de generación de frases.

La herramienta de generación de frases está disponible a través del siguiente enlace través del repositorio "https://github.com/BorjaC/GeneradorDeFrases/tree/master", pulsar "Clone or download" y a continuación "Download ZIP". Una vez descargado el archivo comprimido, descomprimir el archivo y obtener "generador.jar" empleado para la ejecución de la herramienta.

Para la ejecución son necesarios los ficheros:

- *Sintagmas*, fichero que contiene las oraciones correctamente etiquetadas con el número lingüístico (singular o plural) del cual se extraen los sintagmas que serán empleados para crear nuevas frases.

- *Estructuras*, fichero que contiene las estructuras de las oraciones que serán generadas.

Los parámetros para la ejecución serán siempre los siguientes:

- **-est "fichero estructuras":** marca que lo siguiente a leer es el fichero estructuras.
- **-sin "fichero sintagmas":** marca que lo siguiente a leer es el fichero sintagmas.
- **-n "numero entero":** marca el número de oraciones que se van a generar. Este número de oraciones es dividido por el número de estructuras que existe para generar, por lo que es aconsejable emplear múltiplos del número de estructuras.
- **-clave "cadena":** marca la clave para realizar el intercambio de sintagmas. Por ejemplo, NPSUBJ marca que la frase nueva se creara intercambiando el NPSUBJ de la frase del fichero *estructuras* por un NPSUBJ obtenido del fichero *sintagmas* aleatoriamente.

El orden de los parámetros se puede realizar de cualquier forma. A tener en cuenta:

- En todos los casos es necesario emplear todos los parámetros.
- La generación de frases no entiende de concordancia más allá del número, es posible que no exista concordancia semántica. Es responsabilidad de cada cual mantenerla cuidando los sintagmas y las estructuras empleados para el intercambio.
- Si no existen sintagmas dentro de las estructuras que contengan la clave, la generación de nuevas frases a partir de esa estructura no será posible y saltará un error avisando de ello por cada vez que se haya intentado usar esa estructura para generar nuevas frases.

Ejemplos de ejecución:

java -jar generador.jar -est "estructuras" -sin "sintagmas" -n 10 -clave NPSUBJ

La salida esperada son dos ficheros (uno contiene las frases en forma de árbol y la otra las frases en forma de oración) con 10 nuevas frases generadas intercambiando los sintagmas NPSUBJ por otros diferentes.

B Manual del programador

La documentación de la herramienta de validación está disponible mediante el siguiente enlace a través del repositorio "https://github.com/BorjaC/ValidadorSP/tree/master", pulsar "Clone or download" y a continuación "Download ZIP". Una vez descargado el archivo comprimido, descomprimir el archivo y obtener la carpeta "doc" que contiene la documentación en *javadoc* necesario para entender la herramienta.

La documentación de la herramienta de generación de frases está disponible a través del siguiente enlace través del repositorio "https://github.com/BorjaC/GeneradorDeFrases/tree/master", pulsar "Clone or download" y a continuación "Download ZIP". Una vez descargado el archivo comprimido, descomprimir el archivo y obtener la carpeta "doc" que contiene la documentación en *javadoc* necesario para entender la herramienta.

C Resultados y gráficas de aprendizaje del primer experimento.

Para comprobar el correcto funcionamiento, refutar y discutir correctamente los resultados obtenidos se ejecutó la prueba varias veces, así se pudo saber que los picos son producidos por un factor de aleatoriedad.

Supongamos que tenemos un conjunto de 200 frases y que utilizamos una validación cruzada. Para etiquetar tenemos 20 frases por lo tanto el analizador sintáctico ha sido entrenado con 180 oraciones. Claramente en cada oración aparecen una serie de etiquetas, NPSUBJ, VP, PREP, etc., las cuales no tienen por qué aparecer en otras frases.

De este modo si una etiqueta común como VP aparece unas 180 veces en el entrenamiento, una o varias por frase, algunas como NPOBJ1 pueden no aparecer, por lo que el Stanford Parser no tiene forma de saber la existencia de esa etiqueta. Si en las frases de validación, aparece la etiqueta en cuestión entonces se producirá un error las veces que ésta se repita, lo que se traducirá en un peor resultado que si esa etiqueta pareciese.

Además del problema de las etiquetas en esta prueba con el factor de aleatoriedad, aparece el problema de las estructuras. Si al analizador sintáctico se le entrena con unas estructuras determinadas, digamos SUJETO + PREDICADO, esto provoca que una frase con otra estructura, por ejemplo, PREDICADO + SUJETO, resultara mal etiquetada, pues el analizador sintáctico ha aprendido que siempre las frases poseen otra estructura. A continuación se muestras más resultados de la prueba:

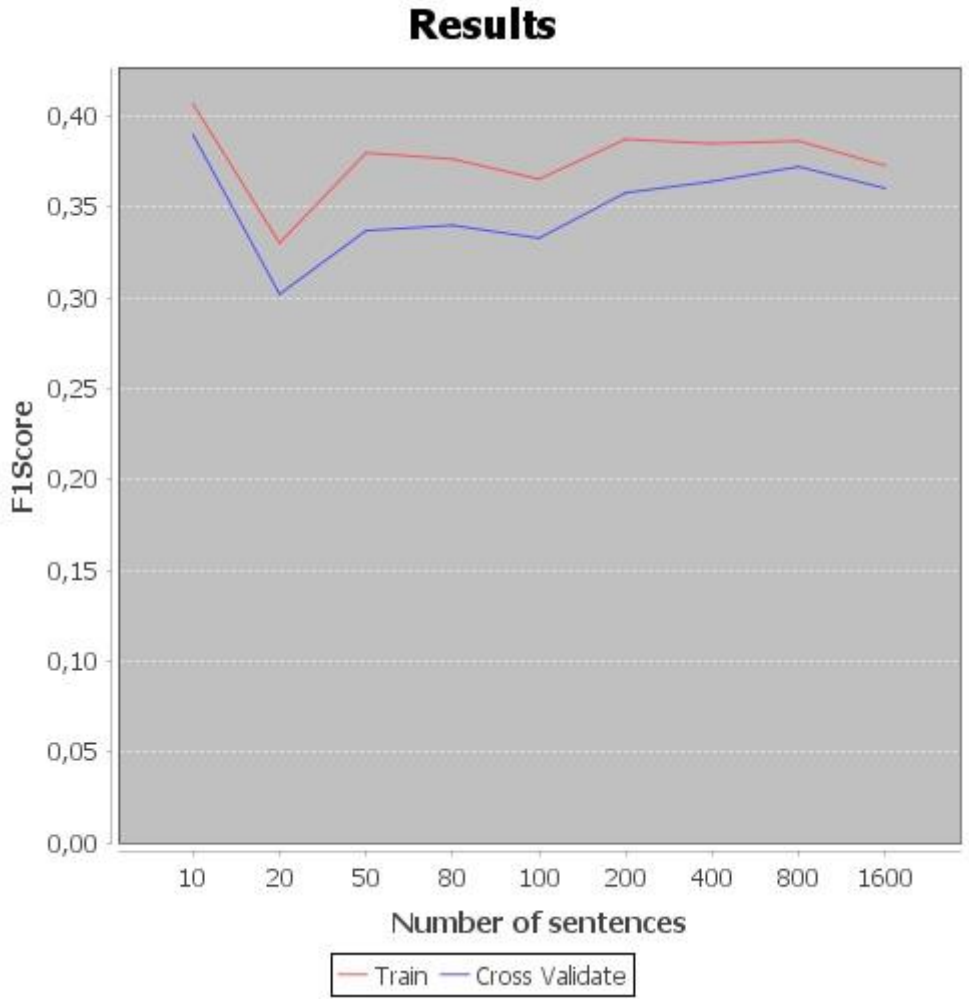
N	*F1Score*	*Precision*	*Recall*
10	0.4063	0.4043	0.4087
20	0.3300	0.3248	0.3354
50	0.3797	0.3691	0.3908
80	0.3763	0.3643	0.3890
100	0.3653	0.3545	0.3766
200	0.3871	0.3765	0.3984
400	0.3848	0.3744	0.3958
800	0.3862	0.3749	0.3981
1600	0.3727	0.3617	0.3844

***Tabla 10*. Resultados del estudio de aprendizaje del analizador sintáctico usando validación simple.**

N	*F1Score*	*Precision*	*Recall*
10	0.3896	0.3903	0.3888
20	0.3019	0.2976	0.3064
50	0.3369	0.3271	0.3473
80	0.3398	0.3272	0.3533
100	0.3327	0.3208	0.3455
200	0.3577	0.3453	0.3711
400	0.3639	0.3521	0.3765
800	0.3721	0.3588	0.3864

1600	0.3602	0.3481	0.3731

***Tabla 11*. Resultados del estudio de aprendizaje del analizador sintáctico usando validación cruzada.**



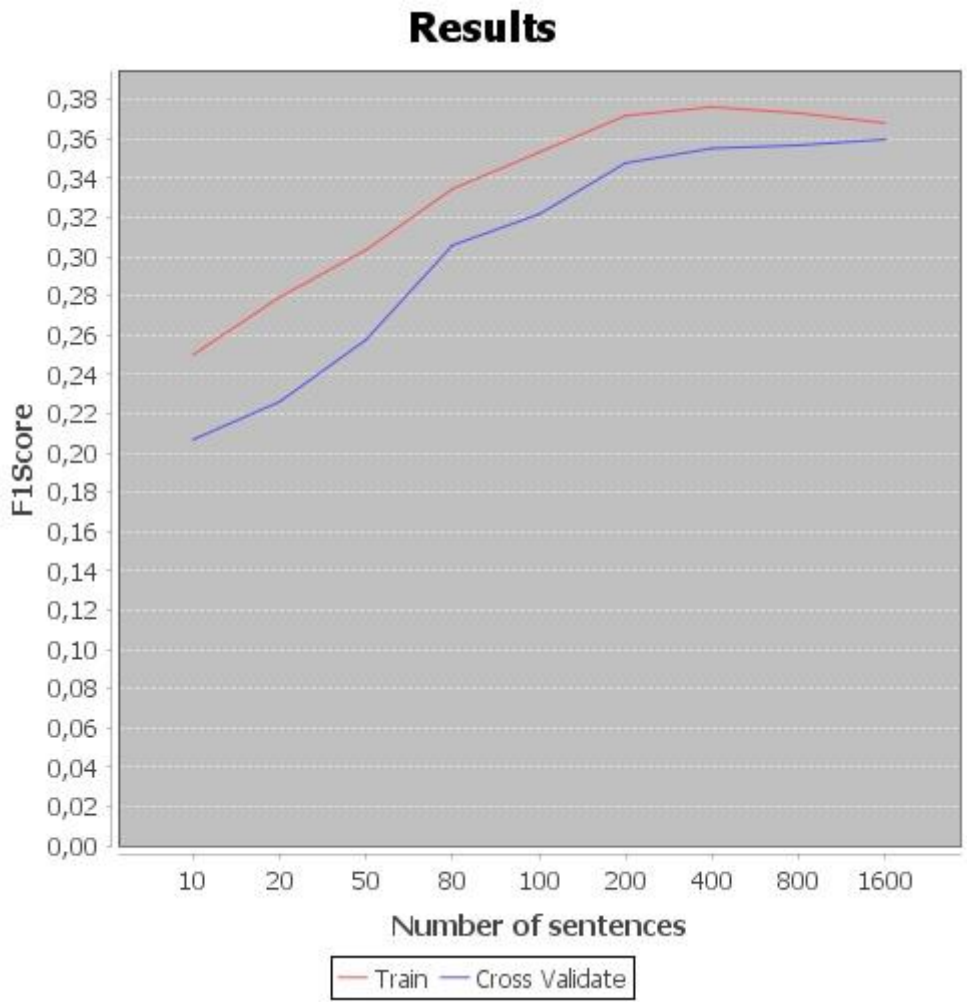
***Gráfica 3*. Curvas de aprendizaje obtenidas en la prueba para analizar la capacidad del aprendizaje del Stanford Parser empleando datos de validación internos. (Segunda ejecución).**

N	*F1Score*	*Precision*	*Recall*
10	0.2500	0.2475	0.2525
20	0.2793	0.2738	0.2851
50	0.3034	0.2942	0.3133
80	0.3345	0.3254	0.3440
100	0.3531	0.3444	0.3623
200	0.3717	0.3605	0.3837
400	0.3761	0.3643	0.3887
800	0.3730	0.3627	0.3839
1600	0.3680	0.3571	0.3795

***Tabla 12.* Resultados del estudio de aprendizaje del Stanford Parser usando validación simple.**

N	*F1Score*	*Precision*	*Recall*
10	0.2078	0.2054	0.2082
20	0.2260	0.2220	0.2302
50	0.2576	0.2478	0.2683
80	0.3058	0.2945	0.3179
100	0.3216	0.3102	0.3337
200	0.3476	0.3348	0.3615
400	0.3551	0.3418	0.3700
800	0.3567	0.3436	0.3708
1600	0.3596	0.3472	0.3730

***Tabla 13*. Resultados del estudio de aprendizaje del Stanford Parser usando validación cruzada.**

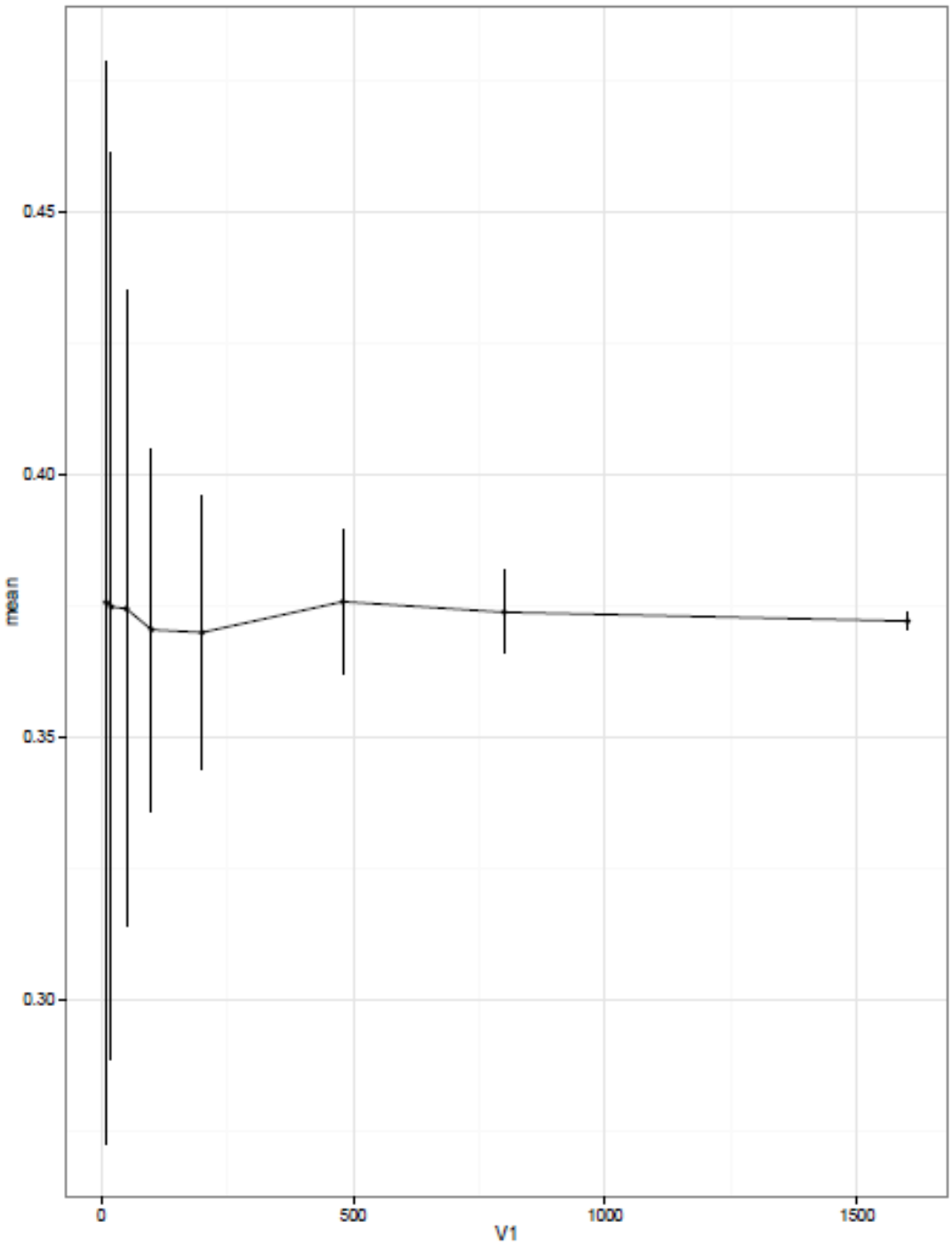


***Gráfica 4*. Curvas de aprendizaje obtenidas en la prueba para analizar la capacidad de aprendizaje del Stanford Parser empleando datos de validación internos. (Tercera ejecución).**

Con estos resultados podemos ver claramente el factor de aleatoriedad presente mencionado anteriormente, el cual es muy importante, sobre todo, con conjunto de frases pequeños. Sin embargo, según aumenta el tamaño del conjunto los resultados, estos tienden

a estabilizarse. Obteniendo las pruebas el mismo resultado al final, pues como era de esperar el mismo conjunto de datos genera el mismo resultado.

La siguiente gráfica muestra los resultados obtenidos en 50 ejecuciones diferentes. Se ve que al principio hay más variación, debido al tamaño del conjunto de datos, pero a medida que este aumenta disminuye la varianza.



***Gráfica 5*. Varianza obtenida en la ejecución de la prueba interna.**

D Frases escogidas para la validación externa.

El/ART presidente/N se/P reunirá/V hoy/ADV para/PREP discutir/V el/ART plan/N ./PUNCT

Dragados/NPR revisan/V sus/POSS políticas/N de/PREP alianzas/N tras/PREP la/ART ruptura/N con/PREP FCC/NPR ./PUNCT

Los/ART chicos/N compran/V gatos/N y/C venden/V perros/N ./PUNCT

Los/DET pulmones/N humanos/ADJ son/V estructuras/N anatómicas/ADJ pertenecientes/ADJ a/PREP el/ART aparato/N respiratorio/ADJ ./PUNCT

Los/DET pulmones/N humanos/ADJ son/V estructuras/N anatómicas/ADJ que/P pertenecen/V a/PREP el/ART aparato/N respiratorio/ADJ ./PUNCT

Los/DET pulmones/N humanos/ADJ son/V estructuras/N anatómicas/ADJ de/PREP origen/N embrionario/ADJ endodérmico/ADJ pertenecientes/ADJ a/PREP el/ART aparato/N respiratorio/ADJ ./PUNCT

Los/DET pulmones/N humanos/ADJ son/V estructuras/N anatómicas/ADJ de/PREP origen/N embrionario/ADJ endodérmico/ADJ que/P pertenecen/V a/PREP el/ART aparato/N respiratorio/ADJ ./PUNCT

El/ART reino/N canta/V muy/ADV bien/ADV ./PUNCT

Oxford/NPR y/C Cambridge/NPR ,/PUNCT las/ART dos/Q joyas/N de/PREP la/ART educación/N universitaria/ADJ británica/ADJ ,/PUNCT ya/ADV no/ADV están/V en/PREP la/ART cima/N de/PREP el/ART mundo/N ./PUNCT

Harvard/NPR ,/PUNCT la/ART universidad/N más/ADV importante/ADJ ,/PUNCT ya/ADV no/ADV está/V en/PREP la/ART mejor/ADJ posición/N ./PUNCT

Harvard/NPR ,/PUNCT una/Q de/PREP las/ART más/ADV importantes/ADJ ,/PUNCT ya/ADV no/ADV está/V en/PREP la/ART mejor/ADJ posición/N ./PUNCT

Pepe/NPR va/V a/PREP el/ART supermercado/N para/PREP comprar/V carne/N ./PUNCT

Informó/V el/ART presidente/N que/C no/ADV habrá/V aumentos/N de/PREP sueldo/N ./PUNCT

Muere/V en/PREP París/NPR el/ART maestro/N de/PREP actores/N Jacques/NPR ./PUNCT

Muere/V el/ART presidente/N de/PREP Francia/N Pepe/NPR ./PUNCT

Vamos/V a/PREP el/ART cine/N ./PUNCT

Lo/P peor/ADJ de/PREP los/ART sueños/N es/V que/C pueden_cumplirse/V ./PUNCT

Existen/V tantos/C caminos/N como/C caminantes/N ./PUNCT

E Resultados de los errores mas frecuentes.

El análisis manual de los datos es muy tedioso, pues son muchas las frases a comparar, para simplificar nos centramos en las frases que obtuvieron una puntuación mayor o igual a 0.9.

En esta comparación descubrimos la generalización de las etiquetas que provoca las opciones de entrenamiento empleadas. Los datos obtenidos de 4 de las 16 frases con dicha puntuación son los siguientes, a la izquierda la salida esperada, a la derecha la salida obtenida:

```
(ROOT
 (S
  (NPSUBJ
   (ART "Una/ART")
   (N "nave/N")
   (ADJP
    (ADJ "interestelar/ADJ")))
  (VP_TENSED
   (V "surca/V")
   (NPOBJ1
    (ART "el/ART")
    (N "cosmos/N")))
  (PUNCT "./PUNCT")))
```

```
(ROOT
 (S
  (NPSUBJ (ART Una) (N nave)
   (ADJP (ADJ interestelar)))
  (VP (V surca)
   (NPOBJ1 (ART el) (N cosmos)))
  (PUNCT .)))
```

```
(ROOT
 (S
  (NPSUBJ
   (NPR "Panamá/NPR"))
  (VP_TENSED
   (V "vivió/V")
   (NPTIME
    (ART "el/ART")
    (NP
     (N "pasado/N"))
    (N "domingo/N"))
   (NPOBJ1
    (ART "una/ART")
    (N "jornada/N")
    (ADJP
     (ADJ "electoral/ADJ"))
    (NP
     (N "cargada/N")
     (PP_DE
      (PREP "de/PREP")
      (NP
       (N "simbolismos/N"))))))
  (PUNCT "./PUNCT")))
```

```
(ROOT
 (S
  (NPSUBJ (NPR Panamá))
  (VP (V vivió)
   (NPTIME (ART el)
    (NP (N pasado))
    (N domingo))
   (NPOBJ1 (ART una) (N jornada)
    (ADJP (ADJ electoral))
    (NP (N cargada)
     (PP (PREP de)
      (NP (N simbolismos))))))
  (PUNCT .)))
```

```
(ROOT
 (S
  (NPSUBJ
   (NPR "Claudia/NPR"))
  (VP_TENSED
   (V "tiene/V")
   (NPOBJ1
    (ART "un/ART")
    (N "trabajo/N")
    (ADJP
     (ADJ "fijo/ADJ"))))
  (PUNCT "./PUNCT")))
```

```
ROOT
 (S
  (NPSUBJ (NPR Claudia))
  (VP (V tiene)
   (NPOBJ1 (ART un) (N trabajo)
    (ADJP (ADJ fijo))))
  (PUNCT .)))
```

```
(ROOT
 (S
  (NPSUBJ
   (POSS "Sus/POSS")
   (ADJP
    (ADJ "grandes/ADJ"))
   (N "virtudes/N"))
  (VP_TENSED
   (V "son/V")
   (NPATTR
    (POSS "sus/POSS")
    (ADJP
     (ADJ "grandes/ADJ"))
    (N "defectos/N")))
  (PUNCT "./PUNCT")))
```

```
(ROOT
 (S
  (NPSUBJ (POSS Sus)
   (ADJP (ADJ grandes))
   (N virtudes))
  (VP (V son)
   (NPATTR (POSS sus)
    (ADJP (ADJ grandes))
    (N defectos)))
  (PUNCT .)))
```

A parte de una estructura más compacta, la cual no influye en la puntuación pues lo que se comparan son etiquetas, se aprecia el patrón de fallo derivado por las etiquetas de VP_TENSED y VP_UNTENSED, que se convierten en una nueva etiqueta que engloba ambas (VP).